THE DL_POLY_2 USER MANUAL

W. Smith, T.R. Forester, I.T. Todorov and M. Leslie

CCLRC Daresbury Laboratory
Daresbury, Warrington WA4 4AD
Cheshire, UK

Version 2.17, December 2006

ABOUT DL_POLY_2

DL_POLY_2 is a parallel molecular dynamics simulation package developed at Daresbury Laboratory by W. Smith and T.R. Forester under the auspices of the Engineering and Physical Sciences Research Council (EPSRC) for the EPSRC's Collaborative Computational Project for the Computer Simulation of Condensed Phases (CCP5) and the Computational Science and Engineering Department at Daresbury Laboratory. The package is the property of the Council for the Central Laboratory of the Research Councils of the United Kingdom.

DL_POLY_2 is issued free **under licence** to academic institutions pursuing scientific research of a non-commercial nature. Commercial organisations may be permitted a licence to use the package after negotiation with the owners. Daresbury Laboratory is the sole centre for distribution of the package. It should not be redistributed to third parties without consent of the owners.

The purpose of the DL_POLY_2 package is to provide software for academic research that is inexpensive, accessible and free of commercial considerations. Users have direct access to source code for modification and inspection. In the spirit of the enterprise, contributions in the form of working code are welcome, provided the code is compatible with DL_POLY_2 in regard to its interfaces and programming style and it is adequately documented.

DISCLAIMER

Neither the CCLRC, EPSRC, CCP5 nor any of the authors of the DL_POLY_2 package or their derivatives guarantee that the packages are free from error. Neither do they accept responsibility for any loss or damage that results from its use.

DL_POLY_2 ACKNOWLEDGEMENTS

DL_POLY_2 was developed under the auspices of the Central Laboratory of the Research Councils, the Engineering and Physical Sciences Research Council, and the former Science and Engineering Research Council, under grants from the Computational Science Initiative and the Science and Materials Computing Committee.

Advice, assistance and encouragement in the development of DL_POLY_2 has been given by many people. We gratefully acknowledge the comments, feedback and bug reports from the CCP5 community in the United Kingdom and throughout the world.

Manual Notation

In the DL_POLY Manual and Reference Manual specific fonts are used to convey specific meanings:

1. *directories* - itallic font indicate unix file directories
2. ROUTINES - small capitals indicate subroutines, functions and programs.
3. *macros* - sloped text indicates a macro (file of unix commands)
4. **directive** - bold text indicates directives or keywords
5. `variables` - typewrite text indicates named variables and parameters
6. FILE - large capitals indicate filenames.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Scope of Chapter

This chapter describes the concept, design and directory structure of DL_POLY_2 and how to obtain a copy of the source code.

1.1 The DL_POLY Package

DL_POLY_2 [1] is a package of subroutines, programs and data files, designed to facilitate molecular dynamics simulations of macromolecules, polymers, ionic systems, solutions and other molecular systems on a distributed memory parallel computer. The package was written to support the UK project CCP5 by Bill Smith and Tim Forester [2] under grants from the Engineering and Physical Sciences Research Council and is the property of the Council for the Central Laboratory of the Research Councils.

Two forms of DL_POLY exist. DL_POLY_2 is the earlier version and is based on a replicated data parallelism. It is suitable for simulations of up to 30,000 atoms on up to 100 processors. DL_POLY_3 is a domain decomposition version, written by I.T. Todorov and W. Smith, and is designed for systems beyond the range of DL_POLY_2 - up to 10,000,000 atoms (and beyond) and 1000 processors. This document is entirely concerned with DL_POLY_2.

Though DL_POLY_2 is designed for distributed memory parallel machines, but we have taken care to ensure that it can, with minimum modification, be run on the popular workstations. Scaling up a simulation from a small workstation to a massively parallel machine is therefore a useful feature of the package.

Users are reminded that we are interested in hearing what other features could be usefully incorporated. We also request that our users respect the copyright of the DL_POLY_2 source and not alter any authorship or copyright notices within. We require that all users of the package register with us, not least because we need to keep everyone abreast of new developments and discovered bugs. We have developed a licence for this purpose, which we hope will ward off litigation (from both sides), without denying access to genuine scientific users.

Further information about the DL_POLY package can be obtained from our website :

http://www.cse.clrc.ac.uk/msi/software/DL_POLY

1.2 Functionality

The following is a list of the features DL_POLY_2 . It is worth reminding users that DL_POLY_2 represents a *package* rather than a single program, so users may consider piecing together their own program with the desired functionality. We will however, supply a consolidated program in the distributed source.

1.2.1 Molecular Systems

DL_POLY_2 will simulate the following molecular species:

1. Simple atomic systems and mixtures e.g. Ne, Ar, Kr, etc.
2. Simple unpolarisable point ions e.g. NaCl, KCl, etc.
3. Polarisable point ions and molecules e.g. MgO, H_2O etc.D
4. Simple rigid molecules e.g. CCl_4, SF_6, Benzene, etc.

5. Rigid molecular ions with point charges e.g. KNO_3, $(NH_4)_2SO_4$, etc.
6. Polymers with rigid bonds e.g. C_nH_{2n+2}
7. Polymers with rigid bonds and point charges e.g. proteins
8. Macromolecules and biological systems
9. Molecules with flexible bonds
10. Silicate glasses and zeolites
11. Simple metals and alloyse.g. Al, Ni, Cu etc.
12. Covalent systems e.g. C, Si, Ge, SiC, SiGe etc.

1.2.2 The DL_POLY_2 Force Field

The DL_POLY_2 force field includes the following features:

1. All common forms of non-bonded atom-atom potential;
2. Atom-atom (site-site) Coulombic potentials;
3. Valence angle potentials;
4. Dihedral angle potentials;
5. Inversion potentials;
6. Improper dihedral angle potentials;
7. 3-body valence angle and hydrogen bond potentials;
8. 4-body inversion potentials;
9. Sutton-Chen density dependent potentials (for metals) [3].
10. The Tersoff density dependent potential for covalent systems [4].

The parameters describing many of these these potentials may be obtained, for example, from the GROMOS [5], Dreiding [6] or AMBER [7] forcefield, which share functional forms. It is relatively easy to adapt DL_POLY_2 to user specific force fields. Note that DL_POLY_2 does not have its own 'official' force field.

1.2.3 Boundary Conditions

DL_POLY_2 will accommodate the following boundary conditions:

1. None e.g. isolated polymer in space.
2. Cubic periodic boundaries.
3. Orthorhombic periodic boundaries.
4. Parallelepiped periodic boundaries.
5. Truncated octahedral periodic boundaries.
6. Rhombic dodecahedral periodic boundaries.
7. Slab (x,y periodic, z nonperiodic).
8. Hexagonal prism periodic boundaries.

These are describe in detail in Appendix B.

1.2.4 The Java Graphical User Interface

DL_POLY_2 has a Graphical User Interface (GUI) written specifically for the package in the Java programming language from Sun microsystems. The Java programming environment is free and it is particularly suitable for building graphical user interfaces. An atractive aspect of java is the portability of the compiled GUI, which may be run without recompiling on any Java supported machine. The GUI is an integral component of the DL_POLY_2 package and is available on exactly the same terms. (See [8].)

1.2.5 Algorithms

1.2.5.1 Parallel Algorithms

DL_POLY_2 exclusively employs the **Replicated Data** parallelisation strategy [9, 10] (see section 2.6.1).

1.2.5.2 Molecular Dynamics Algorithms

The DL_POLY_2 MD algorithms are optionally available in the form of the Verlet Leapfrog or the Velocity Verlet integration algorithms [11].

In the leapfrog scheme a parallel version of the SHAKE algorithm [12, 10] is used for bond constraints and a similar adaptation of the RATTLE algorithm [13] is implmented in the velocity Verlet scheme.

Rigid body rotational motion is handled under the leapfrog scheme with Fincham's implicit quaternion algorithm (FIQA) [14]. For velocity Verlet integration of rigid bodies DL_POLY_2 uses the 'NOSQUISH' algorithm of Miller *et al* [15].

Rigid molecular species linked by rigid bonds are handled with an algorithm of our own devising, called the QSHAKE algorithm [16] which has been adapted for both leapfrog and velocity Verlet schemes.

NVE, NVT, NPT and N$\underline{\underline{\sigma}}$T ensembles are available, with a selection of thermostats and barostats. The velocity Verlet versions are based on the reversible integrators of Martyna *et al* [17].

The NVT algorithms in DL_POLY_2 are those of Evans [18], Berendsen [19]; and Hoover [20]. The NPT algorithms are those of Berendsen [19] and Hoover [20] and the NσT algorithms are those of Berendsen [19] and Hoover [20].

The full range of MD algorithms available in DL_POLY_2 is described in Section 2.5.

1.3 Programming Style

The programming style of DL_POLY_2 is intended to be as uniform as possible. The following stylistic rules apply throughout. Potential contributors of code are requested to note the stylistic convention.

1.3.1 Programming Language

DL_POLY_2 is written exclusively in FORTRAN 90. Use is made of F90 Modules. Explicit type declaration is used throughout.

1.3.2 Memory Management

In DL_POLY_2 , the major array dimensions are calculated at the start of execution and the associated arrays created through the dynamic array allocation features of FORTRAN 90.

1.3.3 Target Computers

DL_POLY_2 is intended for distributed memory parallel computers. However, versions of the program for serial computers are easily produced. To facilitate this all machine specific calls are located in dedicated FORTRAN routines, to permit substitution by appropriate alternatives.

DL_POLY_2 will run on a wide selection of computers. This includes most single processor workstations for which it requires a FORTRAN 90 compiler and (preferably) a UNIX environment. It has also been compiled for a Windows PC using the G95 FORTRAN compiler augmented by the CygWin UNIX shell. The Message Passing Interface (MPI) software is essential for parallel execution.

1.3.4 Version Control System (CVS)

DL_POLY_2 was developed with the aid of the CVS version control system. We strongly recommend that users of DL_POLY_2 adopt this system for local development of the

DL_POLY_2 code, particularly where several users access the same source code. For information on CVS please contact:

$$info - cvs - request@gnu.org$$

or visit the website:

$$http://www.cse.clrc.ac.uk/msi/software/DL\_POLY$$

1.3.5 Required Program Libraries

DL_POLY_2 is, for the most part, self contained and does not require access to additional program libraries. The exception is the MPI software library required for parallel execution.

Users requiring the Smoothed Particle Mesh Ewald (SPME) method may prefer to use a proprietary 3D FFT other than the one (sc dlpfft3) supplied with the package for optimal performance. There are comments in the source code which provide guidance for applications on Cray and IBM computers, which use the routines CCFFT3D and DCFT3 respectively. Similarly users will find comments for the public domain FFT routine FFTWND_FFT.

1.3.6 Internal Documentation

All subroutines are supplied with a header block of FORTRAN COMMENT records giving:

1. The name of the author and/or modifying author
2. The version number or date of production
3. A brief description of the function of the subroutine
4. A copyright statement
5. A CVS revision number and associated data.

Elsewhere FORTRAN COMMENT cards are used liberally.

1.3.7 Subroutine/Function Calling Sequences

The variables in the subroutine arguments are specified in the order:

1. logical and logical arrays
2. character and character arrays
3. integer
4. real and complex
5. integer arrays
6. real and complex arrays

This is admittedly arbitrary, but it really does help with error detection.

1.3.8 FORTRAN Parameters

All global parameters defined by the FORTRAN parameter statements are specified in the module: SETUP_MODULE. All parameters specified in SETUP_MODULE are described by one or more comment cards.

1.3.9 Arithmetic Precision

All real variables and parameters are specified in 64-bit precision (i.e real*8).

1.3.10 Units

Internally all DL_POLY_2 subroutines and functions assume the use of the following defined *molecular units*:

1. The unit of time (t_o) is 1×10^{-12} seconds (i.e. picoseconds).
2. The unit of length (ℓ_o) is 1×10^{-10} metres (i.e. Ångstroms).
3. The unit of mass (m_o) is $1.6605402 \times 10^{-27}$ kilograms (i.e. atomic mass units).
4. The unit of charge (q_o) is $1.60217733 \times 10^{-19}$ coulombs (i.e. unit of proton charge).
5. The unit of energy ($E_o = m_o(\ell_o/t_o)^2$) is $1.6605402 \times 10^{-23}$ Joules (10 J mol^{-1}).
6. The unit of pressure ($\mathcal{P}_o = E_o\ell_o^{-3}$) is 1.6605402×10^{7} Pascal (163.882576 atm).
7. Planck's constant ($\hbar$) which is $6.350780668 \times E_o t_o$.

In addition the following conversion factors are used:
The coulombic conversion factor (γ_o) is:

$$\gamma_o = \frac{1}{E_o}\left[\frac{q_o^2}{4\pi\epsilon_o\ell_o}\right] = 138935.4835$$

such that:

$$U_{MKS} = E_o\gamma_o U_{Internal}$$

Where U represents the configuration energy.
The Boltzmann factor (k_B) is 0.831451115 E_oK^{-1}, such that:

$$T = E_{kin}/k_B$$

represents the conversion from kinetic energy (in internal units) to temperature (in Kelvin).

Note: In the DL_POLY_2 CONTROL and OUTPUT files, the pressure is given in units of kilo-atmospheres (katm) at all times. The unit of energy is either DL_POLY_2 units specified above, or in other units specified by the user at run time. The default is DL_POLY units.

1.3.11 Error Messages

All errors detected by DL_POLY_2 during run time initiate a call to the subroutine ERROR, which prints an error message in the standard output file and terminates the program. All terminations of the program are global (i.e. every node of the parallel computer will be informed of the termination condition and stop executing.)

In addition to terminal error messages, DL_POLY_2 will sometimes print warning messages. These indicate that the code has detected something that is unusual or inconsistent. The detection is non-fatal, but the user should make sure that the warning does represent a harmless condition.

1.4 The DL_POLY_2 Directory Structure

The entire DL_POLY_2 package is stored in a Unix directory structure. The topmost directory is named *dl_poly_2.nn*, where *nn* is a generation number. Beneath this directory are several sub-directories:

sub-directory	contents
srcf90	primary subroutines for the DL_POLY_2 package
utility	subroutines, programs and example data for all utilities
data	example input and output files for DL_POLY_2
bench	large test cases suitable for benchmarking
execute	the DL_POLY_2 run-time directory
build	makefiles to assemble and compile DL_POLY_2 programs
public	directory of routines donated by DL_POLY_2 users
java	directory of Java and FORTRAN routines for the Java GUI

A more detailed description of each sub-directory follows.

1.4.1 The *srcf90* Sub-directory

In this sub-directory all the essential source code for DL_POLY_2 , excluding the utility software. In keeping with the 'package' concept of DL_POLY_2 , it does not contain any complete programs; these are assembled at compile time using an appropriate makefile.

1.4.2 The *utility* Sub-directory

This sub-directory stores all the utility subroutines, functions and programs in DL_POLY_2 , together with examples of data. The various routines in this sub-directory are documented in chapter 6 of this manual. Users who devise their own utilities are advised to store them in the *utility* sub-directory.

1.4.3 The *data* Sub-directory

This sub-directory contains examples of input and output files for testing the released version of DL_POLY_2 . The examples of input data are copied into the *execute* sub-directory when a program is being tested. The test cases are documented in chapter 5.

1.4.4 The *bench* Sub-directory

This directory contains examples of input and output data for DL_POLY_2 that are suitable for benchmarking DL_POLY_2 on large scale computers. These are described in chapter 5.

1.4.5 The *execute* Sub-directory

In the supplied version of DL_POLY_2 , this sub-directory contains only a few macros for copying and storing data from and to the *data* sub-directory and for submitting programs for execution. (These are decribed in section 6.1.1.) However when a DL_POLY_2 program is assembled using its makefile, it will be placed in this sub-directory and will subsequently be executed from here. The output from the job will also appear here, so users will find it convenient to use this sub-directory if they wish to use DL_POLY_2 as intended. (The experienced user is not absolutely required to use DL_POLY_2 this way however.)

1.4.6 The *build* Sub-directory

This sub-directory contains the standard makefiles for the creation (i.e. compilation and linking) of the DL_POLY_2 simulation programs. The makefiles supplied select the appropriate subroutines from the *srcf90* sub-directory and deposit the executable program in the *execute* directory. The user is advised to copy the appropriate makefile into the *srcf90* directory, in case any modifications are required. The copy in the *build* sub-directory will then serve as a backup.

1.4.7 The *public* Sub-directory

This sub-directory contains assorted routines donated by DL_POLY users. Potential users should note that these routines are **unsupported** and come **without any guarantee or liability whatsoever**. They should be regarded as potentially useful resources to be hacked into shape as needed by the user. This directory is available from the CCP5 Program Library by direct FTP(see below).

1.4.8 The *java* Sub-directory

The DL_POLY_2 Java Graphical User Interface (GUI) is based on the Java language developed by Sun. The Java source code for this GUI is to be found in this sub-directory, along with a few FORTRAN sub-sub-directories which contain some additional capabilities accessible from the GUI. These sources are complete and sufficient to create a working GUI,

provided the user has installed the Java Development Kit, (1.4 or above) which is available free from Sun at

$$http://java.sun.com.$$

The GUI, once compiled, may be executed on any machine where Java is installed, though note the FORTRAN components will need to be recompiled if the machine is changed. (See [8].)

1.5 Obtaining the Source Code

To obtain a copy of DL_POLY_2 it is first necessary to log on to the DL_POLY website:

$$http://www.cse.clrc.ac.uk/msi/software/DL\_POLY$$

. Follow the links to the registration page, where you will firstly be shown the DL_POLY software licence, which details the terms and conditions under which the code will be supplied. **By proceeding further with the registration and download process you are signalling your acceptance of the terms of this licence.** Click the 'Registration' button to find the registration page, where you will be invited to enter your name, address and e-mail address. The code is supplied free of charge to academic users, but commercial users will be required to purchase a software licence.

Once the online registration has been completed, information on downloading the DL_POLY_2 source code will be sent by e-mail, so **it is therefore essential to supply a correct e-mail address**.

The *bench* and *public* subdirectories of DL_POLY_2 are not issued in the standard package, but can be downloaded directly from the FTP site (in the ccp5/DL_POLY/DL_POLY_2 directory) as described above.

The DL_POLY_2 User Manual is freely available via the website .

1.6 Other Information

The DL_POLY website:

$$http://www.cse.clrc.ac.uk/msi/software/DL\_POLY$$

provides additional information in the form of

1. Access to all documentation (including licences);
2. Frequently asked questions;
3. Bug reports;
4. Access to the DL_POLY online forum.

Daresbury Laboratory also maintains two DL_POLY_2 associated electronic mailing lists:

1. *dl_poly_news* - to which all registered DL_POLY_2 users are automatically subscribed. It is via this list that error reports and announcements of new versions are made. If you are a DL_POLY_2 user, but not on this list you may request to be added. Contact w.smith@dl.ac.uk.

2. *dl_poly_mail* - is a group list which is available to DL_POLY_2 users by request. Its purpose is to allow DL_POLY_2 users to broadcast information and queries to each other. To subscribe to this list send a mail message to majordomo@dl.ac.uk with the one-line message:

   subscribe dl_poly_mail

   Subsequent messages may be broadcast by e-mailing to the address: dl_poly_mail@dl.ac.uk. Note that this is a vetted list, so electronic spam is not possible.

The DL_POLY **Forum** is a web based centre for all DL_POLY users to exchange comments and queries. You may access the forum through the DL_POLY website. A registration (and vetting) process is required before you can use the forum, but it is open, in principle, to everyone.

Chapter 2

DL_POLY_2 Force Fields and Algorithms

Scope of Chapter

This chapter describes the interaction potentials and simulation algorithms coded into DL_POLY_2 .

2.1 The DL_POLY_2 Force Field

The force field is the set of functions needed to define the interactions in a molecular system. These may have a wide variety of analytical forms, with some basis in chemical physics, which must be parameterised to give the correct energy and forces. A huge variety of forms is possible and for this reason the DL_POLY_2 force field is designed to be adaptable. While it is not supplied with its own force field parameters, many of the functions familiar to GROMOS, [5] Dreiding [6], AMBER [7] and OPLS [21] users have been coded in the package, as well as less familiar forms. In addition DL_POLY_2 retains the possibility of the user defining additional potentials.

In DL_POLY_2 the total configuration energy of a molecular system may be written as:

$$\begin{aligned} U(\underline{r}_1,\underline{r}_2,\ldots,\underline{r}_N) \;=\; & \sum_{i_{bond}=1}^{N_{bond}} U_{bond}(i_{bond},\underline{r}_a,\underline{r}_b) \\ & + \sum_{i_{angle}=1}^{N_{angle}} U_{angle}(i_{angle},\underline{r}_a,\underline{r}_b,\underline{r}_c) \\ & + \sum_{i_{dihed}=1}^{N_{dihed}} U_{dihed}(i_{dihed},\underline{r}_a,\underline{r}_b,\underline{r}_c,\underline{r}_d) \\ & + \sum_{i_{inv}=1}^{N_{inv}} U_{inv}(i_{inv},\underline{r}_a,\underline{r}_b,\underline{r}_c,\underline{r}_d) \\ & + \sum_{i=1}^{N-1}\sum_{j>i}^{N} U_{pair}(i,j,|\underline{r}_i-\underline{r}_j|) \\ & + \sum_{i=1}^{N-2}\sum_{j>i}^{N-1}\sum_{k>j}^{N} U_{3\_body}(i,j,k,\underline{r}_i,\underline{r}_j,\underline{r}_k) \\ & + \sum_{i=1}^{N-1}\sum_{j>i}^{N} U_{Tersoff}(i,j,\underline{r}_i,\underline{r}_j,\underline{R}^N) \\ & + \sum_{i=1}^{N-3}\sum_{j>i}^{N-2}\sum_{k>j}^{N-1}\sum_{n>k}^{N} U_{4\_body}(i,j,k,n,\underline{r}_i,\underline{r}_j,\underline{r}_k,\underline{r}_n) \\ & + \sum_{i=1}^{N} U_{Metal}(i,\underline{r}_i,\underline{R}^N) \\ & + \sum_{i=1}^{N} U_{extn}(i,\underline{r}_i,\underline{v}_i) \end{aligned} \tag{2.1}$$

where U_{bond}, U_{angle}, U_{dihed}, U_{inv}, U_{pair}, U_{3_body}, $U_{Tersoff}$ and U_{4_body} are empirical interaction functions representing chemical bonds, valence angles, dihedral angles, inversion angles, pair-body, three-body, Tersoff (many-body covalent), and four-body forces respectively. The first four are regarded by DL_POLY_2 as *intra*-molecular interactions and the

next five as *inter*-molecular interactions. The term U_{metal} is a density dependent (and therefore many-body) metal potential . The final term U_{extn} represents an *external field* potential.

The position vectors $\underline{r}_a, \underline{r}_b, \underline{r}_c$ and $\underline{r}_d$ refer to the positions of the atoms specifically involved in a given interaction. (Almost universally, it is the *differences* in position that determine the interaction.) A special vector $\underline{R}^N$ is used to indicate a many-body dependence. The numbers N_{bond}, N_{angle}, N_{dihed} and N_{inv} refer to the total numbers of these respective interactions present in the simulated system, and the indices i_{bond}, i_{angle}, i_{inv} and i_{dihed} uniquely specify an individual interaction of each type. It is important to note that there is no global specification of the intramolecular interactions in DL_POLY_2 - all bonds, valence angles and dihedrals must be individually cited.

The indices i, j (and k, n) appearing in the pair-body (and three or four-body) terms indicate the atoms involved in the interaction. There is normally a very large number of these and they are therefore specified according to atom *types* rather than indices. In DL_POLY_2 it is assumed that the pair-body terms arise from van der Waals and/or electrostatic (Coulombic) forces. The former are regarded as short ranged interactions and the latter as long ranged. Long range forces require special techniques to evaluate accurately (see section 2.4.) In DL_POLY_2 the three-body terms are restricted to valence angle and H-bond forms. The nonbonded, three-body, four-body and Tersoff , interactions are globally specified according to the *types* of atoms involved. DL_POLY_2 also has the ability to handle metals via density dependent functions (see below). Though essentially many-body potentials their particular form means they are handled in a manner very similar to pair potentials.

In DL_POLY_2 the intramolecular bonded terms are handled using bookkeeping arrays, which specify the atoms involved in a particular interaction and point to the appropriate arrays of parameters that define the potential. The calculation of bonded forces therefore follows the simple scheme:

1. Every atom in the simulated system is assigned a unique index number from 1 to N;
2. Every intramolecular bonded term U_{type} in the system has a unique index number i_{type}: from 1 to N_{type} where *type* represents a bond, angle or dihedral.
3. A pointer array $key_{type}(n_{type}, i_{type})$ carries the indices of the specific atoms involved in the potential term labelled i_{type}. The dimension n_{type} will be 2, 3 or 4, if the term represents a bond, valence angle, dihedral/inversion.
4. The array $key_{type}(n_{type}, i_{type})$ is used to identify the atoms in a bonded term and the appropriate form of interaction and thus to calculate the energy and forces.

DL_POLY_2 calculates the nonbonded pair interactions using a Verlet neighbour list [11] which is reconstructed at intervals during the simulation. This list records the indices of all 'secondary' atoms within a certain radius of each 'primary' atom; the radius being the cut-off radius (r_{cut}) normally applied to the nonbonded potential function, plus an additional increment (Δr_{cut}). The neighbour list removes the need to scan over all atoms

in the simulation at every timestep. The larger radius ($r_{cut} + \Delta r_{cut}$) means the same list can be used for several timesteps without requiring an update. The frequency at which the list must be updated depends on the thickness of the region Δr_{cut}. DL_POLY_2 has two methods for constructing the neighbour list: the first is based on the Brode-Ahlrichs scheme [22] and is used when r_{cut} is large in comparison with the simulation cell; the second uses the link-cell algorithm [23] when r_{cut} is relatively small. The potential energy and forces arising from the nonbonded interactions are calculated using interpolation tables.

A complication in the construction of the Verlet neighbour list for macromolecules is the concept of *excluded atoms*, which arises from the need to exclude certain atom pairs from the overall list. Which atom pairs need to be excluded is dependent on the precise nature of the force field model, but as a minimum atom pairs linked via extensible bonds or constraints and atoms (grouped in pairs) linked via valence angles are probable candidates. The assumption behind this requirement is that atoms that are formally bonded in a chemical sense, should not participate in nonbonded interactions. (However this is not a universal requirement of all force fields.) The same considerations are needed in dealing with charged excluded atoms. DL_POLY_2 has several subroutines available for constructing the Verlet neighbour list, while taking care of the excluded atoms (see chapter 3 for further information.)

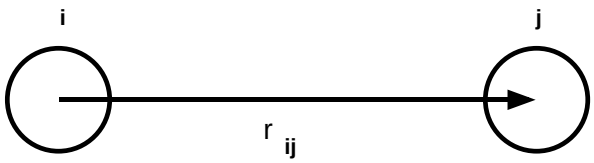
Three- and four-body nonbonded forces are assumed to be short ranged and therefore calculated using the link-cell algorithm [23]. They ignore the possibility of there being any excluded interactions involving the atoms concerned.

Throughout this section the description of the force field assumes the simulated system is described as an assembly of atoms. This is for convenience only and readers should understand that DL_POLY_2 does recognise molecular entities, defined either through constraint bonds or rigid bodies. In the case of rigid bodies, the atomic forces are resolved into molecular forces and torques. These matters are discussed in greater detail later in sections 2.5.2.1 and 2.5.7).

2.2 The Intramolecular Potential Functions

In this section we catalogue and describe the forms of potential function available in DL_POLY_2 The **key words** required to select potential forms are given in brackets () before each definition. The derivations of the atomic forces, virial and stress tensor are also outlined.

2.2.1 Bond Potentials



The interatomic bond vector.

The bond potentials describe *explicit* bonds between specified atoms. They are functions of the interatomic distance only. The potential functions available are as follows.

1. Harmonic bond: (**harm**)
$$U(r_{ij}) = \frac{1}{2}k(r_{ij} - r_o)^2; \tag{2.2}$$

2. Morse potential: (**mors**)
$$U(r_{ij}) = E_o[\{1 - \exp(-k(r_{ij} - r_o))\}^2 - 1]; \tag{2.3}$$

3. 12-6 potential bond: (**12-6**)
$$U(r_{ij}) = \left(\frac{A}{r_{ij}^{12}}\right) - \left(\frac{B}{r_{ij}^{6}}\right); \tag{2.4}$$

4. Restrained harmonic: (**rhrm**)
$$U(r_{ij}) = \frac{1}{2}k(r_{ij} - r_o)^2 \qquad |r_{ij} - r_o| \leq r_c; \tag{2.5}$$
$$U(r_{ij}) = \frac{1}{2}kr_c^2 + kr_c(|r_{ij} - r_o| - r_c) \qquad |r_{ij} - r_o| > r_c; \tag{2.6}$$

5. Quartic potential: (**quar**)
$$U(r_{ij}) = \frac{k}{2}(r_{ij} - r_o)^2 + \frac{k'}{3}(r_{ij} - r_o)^3 + \frac{k''}{4}(r_{ij} - r_o)^4. \tag{2.7}$$

6. Buckingham potential: (**buck**)
$$U(r_{ij}) = A\ \exp\left(-\frac{r_{ij}}{\rho}\right) - \frac{C}{r_{ij}^6}; \tag{2.8}$$

In these formulae r_{ij} is the distance between atoms labelled i and j:
$$r_{ij} = |\underline{r}_j - \underline{r}_i|, \tag{2.9}$$
where $\underline{r}_\ell$ is the position vector of an atom labelled ℓ. [1]

The force on the atom j arising from a bond potential is obtained using the general formula:
$$\underline{f}_j = -\frac{1}{r_{ij}}\left[\frac{\partial}{\partial r_{ij}}U(r_{ij})\right]\underline{r}_{ij}, \tag{2.10}$$
The force $\underline{f}_i$ acting on atom i is the negative of this.

[1] Note: some DL_POLY_2 routines may use the convention that $\underline{r_{ij}} = \underline{r}_i - \underline{r}_j$.

The contribution to be added to the atomic virial is given by

$$\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j, \tag{2.11}$$

with only *one* such contribution from each bond.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.12}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived in this way is symmetric.

In DL_POLY_2 bond forces are handled by the routine BNDFRC.

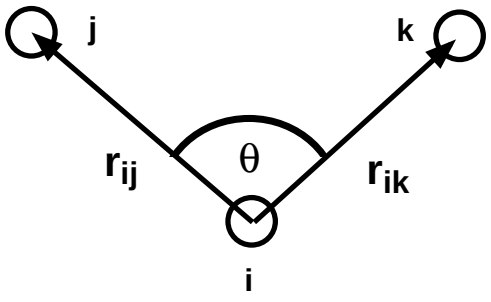
2.2.2 Distance Restraints

In DL_POLY_2 distance restraints, in which the separation between two atoms, is maintained around some preset value r_0 is handled as a special case of bond potentials. As a consequence distance restraints may be applied only between atoms in the same molecule. Unlike with application of the "pure" bond potentials, the electrostatic and van der Waals interactions between the pair of atoms are still evaluated when distance restraints are applied. All the potential forms of the previous section are avaliable as distance restraints, although they have different key words:

1. Harmonic potential: (**-hrm**)
2. Morse potential: (**-mrs**)
3. 12-6 potential bond: (**-126**)
4. Restrained harmonic: (**-rhm**)
5. Quartic potential: (**-qur**)
6. Buckingham potential: (**-bck**)

In DL_POLY_2 distance restraints are handled by the routine BNDFRC.

2.2.3 Valence Angle Potentials



The valence angle and associated vectors

The valence angle potentials describe the bond bending terms between the specified atoms. They should not be confused with the three body potentials described later, which are defined by atom types rather than indices.

1. Harmonic: (**harm**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2; \tag{2.13}$$

2. Quartic: (**quar**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 + \frac{k'}{3}(\theta_{jik} - \theta_0)^3 + \frac{k''}{4}(\theta_{jik} - \theta_0)^4; \tag{2.14}$$

3. Truncated harmonic: (**thrm**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]; \tag{2.15}$$

4. Screened harmonic: (**shrm**)
$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]; \tag{2.16}$$

5. Screened Vessal[24]: (**bvs1**)
$$\begin{aligned} U(\theta_{jik}) &= \frac{k}{8(\theta_{jik} - \pi)^2} \left\{ \left[(\theta_0 - \pi)^2 - (\theta_{jik} - \pi)^2 \right]^2 \right\} \\ &\quad \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]; \end{aligned} \tag{2.17}$$

6. Truncated Vessal[25]: (**bvs2**)
$$\begin{aligned} U(\theta_{jik}) &= k[\theta_{jik}^a(\theta_{jik} - \theta_0)^2(\theta_{jik} + \theta_0 - 2\pi)^2 - \frac{a}{2}\pi^{a-1} \\ &\quad (\theta_{jik} - \theta_0)^2(\pi - \theta_0)^3] \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]. \end{aligned} \tag{2.18}$$

7. Harmonic cosine: (**hcos**)
$$U(\theta_{jik}) = \frac{k}{2}(cos(\theta_{jik}) - cos(\theta_0))^2 \tag{2.19}$$

8. Cosine: (**cos**)
$$U(\theta_{jik}) = A[1 + cos(m\theta_{jik} - \delta)] \tag{2.20}$$

9. MM3 stretch-bend potential (**mmsb**)
$$U(\theta_{jik}) = A(\theta_{jik} - \theta_0)(r_{ij} - r_{ij}^o)(r_{ik} - r_{ik}^o) \tag{2.21}$$

In these formulae θ_{jik} is the angle between bond vectors $\underline{r}_{ij}$ and $\underline{r}_{ik}$:

$$\theta_{jik} = cos^{-1}\left\{\frac{\underline{r}_{ij}\cdot\underline{r}_{ik}}{r_{ij}r_{ik}}\right\} \tag{2.22}$$

In DL_POLY_2 the most general form for the valence angle potentials can be written as:

$$U(\theta_{jik}, r_{ij}, r_{ik}) = A(\theta_{jik})S(r_{ij})S(r_{ik}) \tag{2.23}$$

where $A(\theta)$ is a purely angular function and $S(r)$ is a screening or truncation function. All the function arguments are scalars. With this reduction the force on an atom derived from the valence angle potential is given by:

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha}U(\theta_{jik}, r_{ij}, r_{ik}), \tag{2.24}$$

with atomic label ℓ being one of i, j, k and α indicating the x, y, z component. The derivative is

$$\begin{aligned} -\frac{\partial}{\partial r_\ell^\alpha}U(\theta_{jik}, r_{ij}, r_{ik}) &= -S(r_{ij})S(r_{ik})\frac{\partial}{\partial r_\ell^\alpha}A(\theta_{jik}) \\ &\quad -A(\theta_{jik})S(r_{ik})(\delta_{\ell j}-\delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij}}\frac{\partial}{\partial r_{ij}}S(r_{ij}) \\ &\quad -A(\theta_{jik})S(r_{ij})(\delta_{\ell k}-\delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ik}}\frac{\partial}{\partial r_{ik}}S(r_{ik}), \end{aligned} \tag{2.25}$$

with $\delta_{ab} = 1$ if $a = b$ and $\delta_{ab} = 0$ if $a \neq b$. In the absence of screening terms $S(r)$, this formula reduces to:

$$-\frac{\partial}{\partial r_\ell^\alpha}U(\theta_{jik}, r_{ij}, r_{ik}) = -\frac{\partial}{\partial r_\ell^\alpha}A(\theta_{jik}) \tag{2.26}$$

The derivative of the angular function is

$$-\frac{\partial}{\partial r_\ell^\alpha}A(\theta_{jik}) = \left\{\frac{1}{\sin(\theta_{jik})}\right\}\frac{\partial}{\partial\theta_{jik}}A(\theta_{jik})\frac{\partial}{\partial r_\ell^\alpha}\left\{\frac{\underline{r}_{ij}\cdot\underline{r}_{ik}}{r_{ij}r_{ik}}\right\}, \tag{2.27}$$

with

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha}\left\{\frac{\underline{r}_{ij}\cdot\underline{r}_{ik}}{r_{ij}r_{ik}}\right\} &= (\delta_{\ell j}-\delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ij}r_{ik}} + (\delta_{\ell k}-\delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij}r_{ik}} - \\ &\quad \cos(\theta_{jik})\left\{(\delta_{\ell j}-\delta_{\ell i})\frac{r_{ij}^\alpha}{r_{ij}^2} + (\delta_{\ell k}-\delta_{\ell i})\frac{r_{ik}^\alpha}{r_{ik}^2}\right\} \end{aligned} \tag{2.28}$$

The atomic forces are then completely specified by the derivatives of the particular functions $A(\theta)$ and $S(r)$.

The contribution to be added to the atomic virial is given by

$$\mathcal{W} = -(\underline{r}_{ij}\cdot\underline{f}_j + \underline{r}_{ik}\cdot\underline{f}_k) \tag{2.29}$$

It is worth noting that in the absence of screening terms S(r), the virial is zero [26].

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_{j}^{\beta} + r_{ik}^{\alpha} f_{k}^{\beta} \tag{2.30}$$

and the stress tensor is symmetric.

In DL_POLY_2 valence forces are handled by the routine ANGFRC.

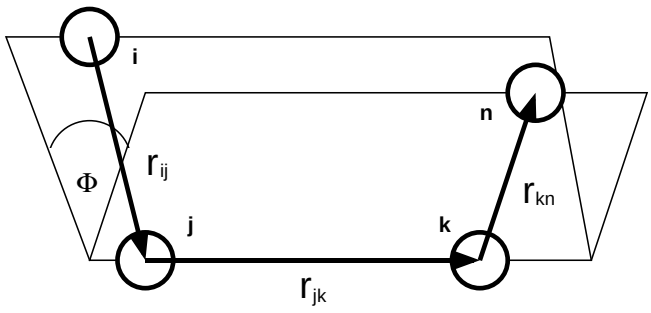
2.2.4 Angular Restraints

In DL_POLY_2 angle restraints, in which the angle subtended by a triplet of atoms, is maintained around some preset value θ_0 is handled as a special case of angle potentials. As a consequence angle restraints may be applied only between atoms in the same molecule. Unlike with application of the "pure" angle potentials, the electrostatic and van der Waals interactions between the pair of atoms are still evaluated when distance restraints are applied. All the potential forms of the previous section are avaliable as angular restraints, although they have different key words:

1. Harmonic: (**-hrm**)
2. Quartic: (**-qur**)
3. Truncated harmonic: (**-thm**)
4. Screened harmonic: (**-shm**)
5. Screened Vessal[24]: (**-bv1**)
6. Truncated Vessal[25]: (**-bv2**)
7. Harmonic cosine: (**-hcs**)
8. Cosine : (**-cos**)
9. MM3 stretch-bend: (**-msb**)

In DL_POLY_2 angular restraints are handled by the routine ANGFRC.

2.2.5 Dihedral Angle Potentials



The dihedral angle and associated vectors

The dihedral angle potentials describe the interaction arising from torsional forces in molecules. (They are sometimes referred to as torsion potentials.) They require the specification of four atomic positions. The potential functions available in DL_POLY_2 are as follows.

1. Cosine potential: (**cos**)

$$U(\phi_{ijkn}) = A\left[1 + \cos(m\phi_{ijkn} - \delta)\right] \tag{2.31}$$

2. Harmonic: (**harm**)

$$U(\phi_{ijkn}) = \frac{1}{2}k(\phi_{ijkn} - \phi_0)^2 \tag{2.32}$$

3. Harmonic cosine: (**hcos**)

$$U(\phi_{ijkn}) = \frac{k}{2}(cos(\phi_{ijkn}) - cos(\phi_0))^2 \tag{2.33}$$

4. Triple cosine: (**cos3**)

$$U(\phi) = \frac{1}{2}A_1(1 + cos(\phi)) + \frac{1}{2}A_2(1 - cos(2\phi)) + \frac{1}{2}A_3(1 + cos(3\phi)) \tag{2.34}$$

5. Ryckaert-Bellemans hydrocarbon potential: (**ryck**)

$$U(\phi_{ijkn}) = A(a_0 + \sum_{i=1}^{5}(a_i cos^i(\phi)) \tag{2.35}$$

6. Ryckaert-Bellemans fluorinated potential: (**rbf**)

$$U(\phi_{ijkn}) = B(b_0 + \sum_{i=1}^{5}(b_i cos^i(\phi)) \tag{2.36}$$

7. OPLS angle potential

$$U(\phi_{ijkn}) = a_0 + 0.5 * (a_1(1 + cos(\phi)) + a_2(1 - cos(2\phi)) + a_3(1 + cos(3\phi))) \tag{2.37}$$

In these formulae ϕ_{ijkn} is the dihedral angle defined by

$$\phi_{ijkn} = \cos^{-1}\{B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn})\}, \tag{2.38}$$

with

$$B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn}) = \left\{ \frac{(\underline{r}_{ij} \times \underline{r}_{jk}) \cdot (\underline{r}_{jk} \times \underline{r}_{kn})}{|\underline{r}_{ij} \times \underline{r}_{jk}||\underline{r}_{jk} \times \underline{r}_{kn}|} \right\}. \tag{2.39}$$

With this definition, the sign of the dihedral angle is positive if the vector product $(\underline{r}_{ij} \times \underline{r}_{jk}) \times (\underline{r}_{jk} \times \underline{r}_{kn})$ is in the same direction as the bond vector $\underline{r}_{jk}$ and negative if in the opposite direction.

The force on an atom arising from the dihedral potential is given by

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha} U(\phi_{ijkn}), \tag{2.40}$$

with ℓ being one of i, j, k, n and α one of x, y, z. This may be expanded into

$$-\frac{\partial}{\partial r_\ell^\alpha} U(\phi_{ijkn}) = \left\{ \frac{1}{\sin(\phi_{ijkn})} \right\} \frac{\partial}{\partial \phi_{ijkn}} U(\phi_{ijkn}) \frac{\partial}{\partial r_\ell^\alpha} B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn}). \tag{2.41}$$

The derivative of the function $B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn})$ is

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} B(\underline{r}_{ij}, \underline{r}_{jk}, \underline{r}_{kn}) = & \frac{1}{|\underline{r}_{ij} \times \underline{r}_{jk}||\underline{r}_{jk} \times \underline{r}_{kn}|} \frac{\partial}{\partial r_\ell^\alpha} \{(\underline{r}_{ij} \times \underline{r}_{jk}) \cdot (\underline{r}_{jk} \times \underline{r}_{kn})\} \\ & - \frac{\cos(\phi_{ijkn})}{2} \left\{ \frac{1}{|\underline{r}_{ij} \times \underline{r}_{jk}|^2} \frac{\partial}{\partial r_\ell^\alpha} |\underline{r}_{ij} \times \underline{r}_{jk}|^2 + \frac{1}{|\underline{r}_{jk} \times \underline{r}_{kn}|^2} \frac{\partial}{\partial r_\ell^\alpha} |\underline{r}_{jk} \times \underline{r}_{kn}|^2 \right\}, \end{aligned} \tag{2.42}$$

with

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} \{(\underline{r}_{ij} \times \underline{r}_{jk}) \cdot (\underline{r}_{jk} \times \underline{r}_{kn})\} = & \; r_{ij}^\alpha ([\underline{r}_{jk} \underline{r}_{jk}]_\alpha (\delta_{\ell k} - \delta_{\ell n}) + [\underline{r}_{jk} \underline{r}_{kn}]_\alpha (\delta_{\ell k} - \delta_{\ell j})) + \\ & r_{jk}^\alpha ([\underline{r}_{ij} \underline{r}_{jk}]_\alpha (\delta_{\ell n} - \delta_{\ell k}) + [\underline{r}_{jk} \underline{r}_{kn}]_\alpha (\delta_{\ell j} - \delta_{\ell i})) + \\ & r_{kn}^\alpha ([\underline{r}_{ij} \underline{r}_{jk}]_\alpha (\delta_{\ell k} - \delta_{\ell j}) + [\underline{r}_{jk} \underline{r}_{jk}]_\alpha (\delta_{\ell i} - \delta_{\ell j})) + \\ & 2 r_{jk}^\alpha [\underline{r}_{ij} \underline{r}_{kn}]_\alpha (\delta_{\ell j} - \delta_{\ell k}), \end{aligned} \tag{2.43}$$

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} |\underline{r}_{ij} \times \underline{r}_{jk}|^2 = & \; 2 r_{ij}^\alpha ([\underline{r}_{jk} \underline{r}_{jk}]_\alpha (\delta_{\ell j} - \delta_{\ell i}) + [\underline{r}_{ij} \underline{r}_{jk}]_\alpha (\delta_{\ell j} - \delta_{\ell k})) + \\ & 2 r_{jk}^\alpha ([\underline{r}_{ij} \underline{r}_{ij}]_\alpha (\delta_{\ell k} - \delta_{\ell j}) + [\underline{r}_{ij} \underline{r}_{jk}]_\alpha (\delta_{\ell i} - \delta_{\ell j})), \end{aligned} \tag{2.44}$$

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} |\underline{r}_{jk} \times \underline{r}_{kn}|^2 = & \; 2 r_{kn}^\alpha ([\underline{r}_{jk} \underline{r}_{jk}]_\alpha (\delta_{\ell n} - \delta_{\ell k}) + [\underline{r}_{jk} \underline{r}_{kn}]_\alpha (\delta_{\ell j} - \delta_{\ell k})) + \\ & 2 r_{jk}^\alpha ([\underline{r}_{kn} \underline{r}_{kn}]_\alpha (\delta_{\ell k} - \delta_{\ell j}) + [\underline{r}_{jk} \underline{r}_{kn}]_\alpha (\delta_{\ell k} - \delta_{\ell n})). \end{aligned} \tag{2.45}$$

Where we have used the the following definition:

$$[\underline{a}\ \underline{b}]_\alpha = \sum_\beta (1 - \delta_{\alpha\beta}) a^\beta b^\beta. \tag{2.46}$$

Formally, the contribution to be added to the atomic virial is given by

$$\mathcal{W} = -\sum_{i=1}^{4} \underline{r}_i \cdot \underline{f}_i \tag{2.47}$$

However it is possible to show (by tedious algebra using the above formulae, or more elegantly by thermodynamic arguments [26],) that the dihedral makes *no* contribution to the atomic virial.

The contribution to be added to the atomic stress tensor is given by

$$\begin{aligned} \sigma^{\alpha\beta} &= r_{ij}^{\alpha} p_i^{\beta} + r_{jk}^{\alpha} p_{jk}^{\beta} + r_{kn}^{\alpha} p_n^{\beta} \\ &\quad - \frac{\cos(\phi_{ijkn})}{2} \left\{ r_{ij}^{\alpha} g_i^{\beta} + r_{jk}^{\alpha} g_k^{\beta} + r_{jk}^{\alpha} h_j^{\beta} + r_{kn}^{\alpha} h_n^{\beta} \right\}, \end{aligned} \tag{2.48}$$

with

$$p_i^{\alpha} = (r_{jk}^{\alpha}[\underline{r}_{jk}\underline{r}_{kn}]_{\alpha} - r_{kn}^{\alpha}[\underline{r}_{jk}\underline{r}_{jk}]_{\alpha})/(|\underline{r}_{ij} \times \underline{r}_{jk}||\underline{r}_{jk} \times \underline{r}_{kn}|) \tag{2.49}$$

$$p_n^{\alpha} = (r_{jk}^{\alpha}[\underline{r}_{ij}\underline{r}_{jk}]_{\alpha} - r_{ij}^{\alpha}[\underline{r}_{jk}\underline{r}_{jk}]_{\alpha})/(|\underline{r}_{ij} \times \underline{r}_{jk}||\underline{r}_{jk} \times \underline{r}_{kn}|) \tag{2.50}$$

$$p_{jk}^{\alpha} = (r_{ij}^{\alpha}[\underline{r}_{jk}\underline{r}_{kn}]_{\alpha} + r_{kn}^{\alpha}[\underline{r}_{ij}\underline{r}_{jk}]_{\alpha} - 2r_{jk}^{\alpha}[\underline{r}_{ij}\underline{r}_{kn}]_{\alpha})/(|\underline{r}_{ij} \times \underline{r}_{jk}||\underline{r}_{jk} \times \underline{r}_{kn}|) \tag{2.51}$$

$$g_i^{\alpha} = 2(r_{ij}^{\alpha}[\underline{r}_{jk}\underline{r}_{jk}]_{\alpha} - r_{jk}^{\alpha}[\underline{r}_{ij}\underline{r}_{jk}]_{\alpha})/|\underline{r}_{ij} \times \underline{r}_{jk}|^2 \tag{2.52}$$

$$g_k^{\alpha} = 2(r_{jk}^{\alpha}[\underline{r}_{ij}\underline{r}_{ij}]_{\alpha} - r_{ij}^{\alpha}[\underline{r}_{ij}\underline{r}_{jk}]_{\alpha})/|\underline{r}_{ij} \times \underline{r}_{jk}|^2 \tag{2.53}$$

$$h_j^{\alpha} = 2(r_{jk}^{\alpha}[\underline{r}_{kn}\underline{r}_{kn}]_{\alpha} - r_{kn}^{\alpha}[\underline{r}_{jk}\underline{r}_{kn}]_{\alpha})/|\underline{r}_{jk} \times \underline{r}_{kn}|^2 \tag{2.54}$$

$$h_n^{\alpha} = 2(r_{kn}^{\alpha}[\underline{r}_{kn}\underline{r}_{kn}]_{\alpha} - r_{jk}^{\alpha}[\underline{r}_{jk}\underline{r}_{kn}]_{\alpha})/|\underline{r}_{jk} \times \underline{r}_{kn}|^2 \tag{2.55}$$

The sum of the diagonal elements of the stress tensor is zero (since the virial is zero) and the matrix is symmetric.

Lastly, it should be noted that the above description does not take into account the possible inclusion of distance-dependent 1-4 interactions, as permitted by some force fields. Such interactions are permissible in DL_POLY_2 and are described in the section on pair potentials below. DL_POLY_2 also permits scaling of the 1-4 interactions by a numerical factor. 1-4 interactions do, of course, contribute to the atomic virial.

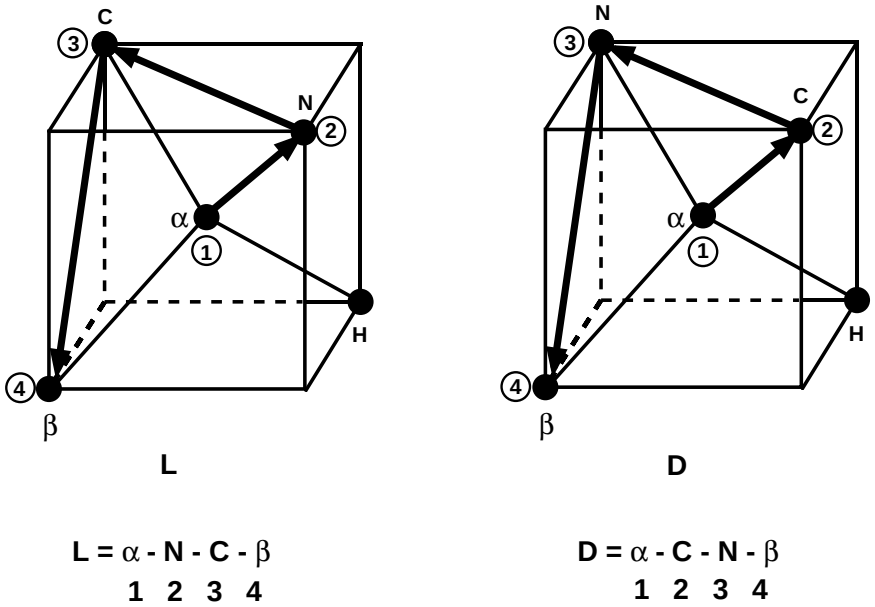
In DL_POLY_2 dihedral forces are handled by the routine DIHFRC.

2.2.6 Improper Dihedral Angle Potentials

Improper dihedrals are used to restrict the geometry of molecules and as such need not have a simple relation to conventional chemical bonding. DL_POLY_2 makes no distinction between dihedral angle functions and improper dihedrals (both are calculated by the same subroutines) and all the comments made in the preceeding section apply.

An important example of the use of the improper dihedral is to conserve the structure of chiral centres in molecules modelled by united-atom centres. For example α-amino acids such as alanine ($CH_3CH(NH_2)COOH$), in which it is common to represent the CH_3 and CH groups as single centres. Conservation of the chirality of the α carbon is achieved by defining a harmonic improper dihedral angle potential with an equilibrium angle of

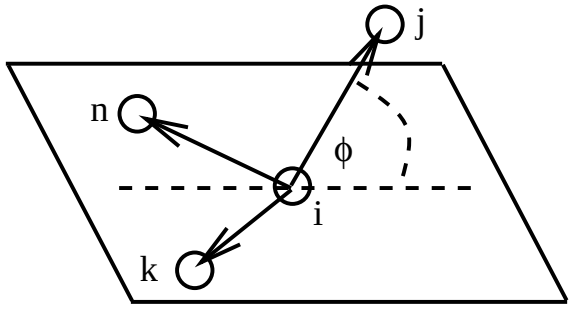
35.264^o. The angle is defined by vectors $\underline{r}_{12}$, $\underline{r}_{23}$ and $\underline{r}_{34}$, where the atoms 1,2,3 and 4 are shown in the following figure. The figure defines the D and L enantiomers consistent with the international (IUPAC) convention. When defining the dihedral, the atom indices are entered in DL_POLY_2 in the order 1-2-3-4.



The L and D enantiomers and defining vectors

In DL_POLY_2 improper dihedral forces are handled by the routine DIHFRC.

2.2.7 Inversion Angle Potentials



The inversion angle and associated vectors

The inversion angle potentials describe the interaction arising from a particular geometry of three atoms around a central atom. The best known example of this is the

arrangement of hydrogen atoms around nitrogen in ammonia to form a trigonal pyramid. The hydrogens can 'flip' like an inverting umbrella to an alternative structure, which in this case is identical, but in principle causes a change in chirality. The force restraining the ammonia to one structure can be described as an inversion potential (though it is usually augmented by valence angle potentials also). The inversion angle is defined in the figure above - **note that the inversion angle potential is a sum of the three possible inversion angle terms.** It resembles a dihedral potential in that it requires the specification of four atomic positions.

The potential functions available in DL_POLY_2 are as follows.

1. Harmonic: (**harm**)
$$U(\phi_{ijkn}) = \frac{1}{2}k(\phi_{ijkn} - \phi_0)^2 \tag{2.56}$$

2. Harmonic cosine: (**hcos**)
$$U(\phi_{ijkn}) = \frac{k}{2}(cos(\phi_{ijkn}) - cos(\phi_0))^2 \tag{2.57}$$

3. Planar potential: (**plan**)
$$U(\phi_{ijkn}) = A\,[1 - \cos(\phi_{ijkn})] \tag{2.58}$$

In these formulae ϕ_{ijkn} is the inversion angle defined by

$$\phi_{ijkn} = \cos^{-1}\left\{\frac{\underline{r}_{ij}\cdot\underline{w}_{kn}}{r_{ij}w_{kn}}\right\}, \tag{2.59}$$

with

$$\underline{w}_{kn} = (\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{\underline{u}}_{kn} + (\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{\underline{v}}_{kn} \tag{2.60}$$

and the unit vectors

$$\begin{aligned}\hat{\underline{u}}_{kn} &= (\hat{\underline{r}}_{ik} + \hat{\underline{r}}_{in})/|\hat{\underline{r}}_{ik} + \hat{\underline{r}}_{in}| \\ \hat{\underline{v}}_{kn} &= (\hat{\underline{r}}_{ik} - \hat{\underline{r}}_{in})/|\hat{\underline{r}}_{ik} - \hat{\underline{r}}_{in}|.\end{aligned} \tag{2.61}$$

As usual, $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$ *etc.* and the hat $\hat{\underline{r}}$ indicates a *unit* vector in the direction of $\underline{r}$. The total inversion potential requires the calculation of three such angles, the formula being derived from the above using the cyclic permutation of the indices $j \to k \to n \to j$ *etc.*

Equivalently, the angle ϕ_{ijkn} may be written as

$$\phi_{ijkn} = \cos^{-1}\left\{\frac{[(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})^2 + (\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})^2]^{1/2}}{r_{ij}}\right\} \tag{2.62}$$

Formally, the force on an atom arising from the inversion potential is given by

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha}U(\phi_{ijkn}), \tag{2.63}$$

with ℓ being one of i, j, k, n and α one of x, y, z. This may be expanded into

$$-\frac{\partial}{\partial r_\ell^\alpha}U(\phi_{ijkn}) = \left\{\frac{1}{\sin(\phi_{ijkn})}\right\}\frac{\partial}{\partial\phi_{ijkn}}U(\phi_{ijkn})\times \frac{\partial}{\partial r_\ell^\alpha}\left\{\frac{[(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})^2+(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})^2]^{1/2}}{r_{ij}}\right\}. \tag{2.64}$$

Following through the (extremely tedious!) differentiation gives the result:

$$\begin{aligned} f_\ell^\alpha = & \left\{\frac{1}{\sin(\phi_{ijkn})}\right\}\frac{\partial}{\partial\phi_{ijkn}}U(\phi_{ijkn})\times \\ & \left\{-(\delta_{\ell j}-\delta_{\ell i})\frac{cos(\phi_{ijkn})}{r_{ij}^2}r_{ij}^\alpha+\frac{1}{r_{ij}w_{kn}}\left[(\delta_{\ell j}-\delta_{\ell i})\{(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{u}_{kn}^\alpha+(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{v}_{kn}^\alpha\}\right.\right. \\ & +(\delta_{\ell k}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn}}{u_{kn}r_{ik}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{u}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{ik}-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})(\underline{r}_{ik}\cdot\hat{\underline{u}}_{kn}))\frac{r_{ik}^\alpha}{r_{ik}^2}\right\} \\ & +(\delta_{\ell k}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn}}{v_{kn}r_{ik}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{v}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{ik}-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})(\underline{r}_{ik}\cdot\hat{\underline{v}}_{kn}))\frac{r_{ik}^\alpha}{r_{ik}^2}\right\} \\ & +(\delta_{\ell n}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn}}{u_{kn}r_{in}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})\hat{u}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{in}-(\underline{r}_{ij}\cdot\hat{\underline{u}}_{kn})(\underline{r}_{in}\cdot\hat{\underline{u}}_{kn}))\frac{r_{in}^\alpha}{r_{in}^2}\right\} \\ & \left.\left.+(\delta_{\ell n}-\delta_{\ell i})\frac{\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn}}{v_{kn}r_{in}}\left\{r_{ij}^\alpha-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})\hat{v}_{kn}^\alpha-(\underline{r}_{ij}\cdot\underline{r}_{in}-(\underline{r}_{ij}\cdot\hat{\underline{v}}_{kn})(\underline{r}_{in}\cdot\hat{\underline{v}}_{kn}))\frac{r_{in}^\alpha}{r_{in}^2}\right\}\right]\right\} \end{aligned} \tag{2.65}$$

This general formula applies to all atoms $\ell = i, j, k, n$. It must be remembered however, that these formulae apply to just one of the three contributing terms (i.e. one angle ϕ) of the full inversion potential: specifically the inversion angle pertaining to the out-of-plane vector $\underline{r}_{ij}$. The contributions arising from the other vectors $\underline{r}_{ik}$ and $\underline{r}_{in}$ are obtained by the cyclic permutation of the indices in the manner described above. All these force contributions must be added to the final atomic forces.

Formally, the contribution to be added to the atomic virial is given by

$$\mathcal{W} = -\sum_{i=1}^{4}\underline{r}_i\cdot\underline{f}_i \tag{2.66}$$

However it is possible to show by thermodynamic arguments (*cf* [26],) or simply from the fact that the sum of forces on atoms j,k and n is equal and opposite to the force on atom i, that the inversion potential makes *no* contribution to the atomic virial.

If the force components f_ℓ^α for atoms $\ell = i, j, k, n$ are calculated using the above formulae, it is easily seen that the contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^\alpha f_j^\beta + r_{ik}^\alpha f_k^\beta + r_{in}^\alpha f_n^\beta \tag{2.67}$$

The sum of the diagonal elements of the stress tensor is zero (since the virial is zero) and the matrix is symmetric.

In DL_POLY_2 inversion forces are handled by the routine INVFRC.

2.2.8 Tethering Forces

DL_POLY_2 also allows atomic sites to be tethered to a fixed point in space, $\underline{r}_0$ taken as their position at the beginning of the simulation. This is also known as position restraining. The specification, which comes as part of the molecular description, requires a tether potential type and the associated interaction parameters.

Note, firstly, that application of tethering potentials means that momentum will no longer be a conserved quantity of the simulation. Secondly, in constant pressure simulations, where the MD cell changes size or shape, the reference position is scaled with the cell vectors.

The potential functions available in DL_POLY_2 are as follows, in each case r_{i0} is the distance of the atom from its position at $t = 0$:

1. harmonic potential: (**harm**)
$$U(r_{i0}) = \frac{1}{2}k(r_{i0})^2; \tag{2.68}$$

2. restrained harmonic :(**rhrm**)
$$U(r_{i0}) = \frac{1}{2}k(r_{i0})^2 \qquad r_{i0} \leq r_c; \tag{2.69}$$
$$U(r_{i0}) = \frac{1}{2}kr_c^2 + kr_c(r_{i0} - r_c) \qquad r_{i0} > r_c; \tag{2.70}$$

3. Quartic potential: (**quar**)
$$U(r_{i0}) = \frac{k}{2}(r_{i0})^2 + \frac{k'}{3}(r_{i0})^3 + \frac{k''}{4}(r_{i0})^4. \tag{2.71}$$

The force on the atom i arising from a tether potential is obtained using the general formula:
$$\underline{f}_i = -\frac{1}{r_{i0}}\left[\frac{\partial}{\partial r_{i0}}U(r_{i0})\right]\underline{r}_{i0}, \tag{2.72}$$

The contribution to be added to the atomic virial is given by
$$\mathcal{W} = \underline{r}_{i0} \cdot \underline{f}_i, \tag{2.73}$$

The contribution to be added to the atomic stress tensor is given by
$$\sigma^{\alpha\beta} = -r_{i0}^{\alpha} f_i^{\beta}, \tag{2.74}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived in this way is symmetric.

In DL_POLY_2 bond forces are handled by the routine TETHFRC.

2.2.9 Frozen Atoms

DL_POLY_2 also allows atoms to be completely immobilised (*i.e.* "frozen" at a fixed point in the MD cell). This is achieved by setting all forces and velocities associated with that atom to zero during each MD timestep. Frozen atoms are signalled by assigning an atom a non-zero value for the freeze parameter in the FIELD file. DL_POLY_2 does not calculate contributions to the virial or the stress tensor arising from the constraints required to freeze atomic positions. In DL_POLY_2 the frozen atom option cannot be used for sites in a rigid body. As with the tethering potential, the reference position is scaled with the cell vectors in constant pressure simulations.

In DL_POLY_2 the frozen atom option is handled by the subroutine FREEZE.

2.3 The Intermolecular Potential Functions

In this section we outline the pair-body, three-body and four-body potential functions available in DL_POLY_2 . An important distinction between these and intramolecular (bond) forces in DL_POLY_2 is that they are specified by *atom types* rather than atom indices.

2.3.1 Short Ranged (van der Waals) Potentials

The short ranged pair forces available in DL_POLY_2 are as follows.

1. 12 - 6 potential: (**12-6**)
$$U(r_{ij}) = \left(\frac{A}{r_{ij}^{12}}\right) - \left(\frac{B}{r_{ij}^{6}}\right); \tag{2.75}$$

2. Lennard-Jones: (**lj**)
$$U(r_{ij}) = 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]; \tag{2.76}$$

3. n - m potential [27]: (**nm**)
$$U(r_{ij}) = \frac{E_o}{(n-m)}\left[m\left(\frac{r_o}{r_{ij}}\right)^{n} - n\left(\frac{r_o}{r_{ij}}\right)^{m}\right]; \tag{2.77}$$

4. Buckingham potential: (**buck**)
$$U(r_{ij}) = A\ \exp\left(-\frac{r_{ij}}{\rho}\right) - \frac{C}{r_{ij}^{6}}; \tag{2.78}$$

5. Born-Huggins-Meyer potential: (**bhm**)
$$U(r_{ij}) = A\ \exp[B(\sigma - r_{ij})] - \frac{C}{r_{ij}^{6}} - \frac{D}{r_{ij}^{8}}; \tag{2.79}$$

6. Hydrogen-bond (12 - 10) potential: (**hbnd**)

$$U(r_{ij}) = \left(\frac{A}{r_{ij}^{12}}\right) - \left(\frac{B}{r_{ij}^{10}}\right); \tag{2.80}$$

7. Shifted force n - m potential [27]: (**snm**)

$$\begin{aligned} U(r_{ij}) &= \frac{\alpha E_o}{(n-m)}\left[m\beta^n\left\{\left(\frac{r_o}{r_{ij}}\right)^n - \left(\frac{1}{\gamma}\right)^n\right\} - n\beta^m\left\{\left(\frac{r_o}{r_{ij}}\right)^m - \left(\frac{1}{\gamma}\right)^m\right\}\right] \\ &+\frac{nm\alpha E_o}{(n-m)}\left(\frac{r_{ij}-\gamma r_o}{\gamma r_o}\right)\left\{\left(\frac{\beta}{\gamma}\right)^n - \left(\frac{\beta}{\gamma}\right)^m\right\} \end{aligned} \tag{2.81}$$

with

$$\gamma = \frac{r_{cut}}{r_o} \tag{2.82}$$

$$\beta = \gamma\left(\frac{\gamma^{m+1}-1}{\gamma^{n+1}-1}\right)^{\frac{1}{n-m}} \tag{2.83}$$

$$\alpha = \frac{(n-m)}{[n\beta^m(1+(m/\gamma-m-1)/\gamma^m) - m\beta^n(1+(n/\gamma-n-1)/\gamma^n)]} \tag{2.84}$$

This peculiar form has the advantage over the standard shifted n-m potential in that both E_o and r_0 (well depth and location of minimum) retain their original values after the shifting process.

8. Morse potential: (**mors**)

$$U(r_{ij}) = E_o[\{1-\exp(-k(r_{ij}-r_o))\}^2 - 1]; \tag{2.85}$$

9. Tabulation: (**tab**). The potential is defined numerically only.

The parameters defining these potentials are supplied to DL_POLY_2 at run time (see the description of the FIELD file in section 4.1.3). Each atom type in the system is specified by a unique eight-character label defined by the user. The pair potential is then defined internally by the combination of two atom labels.

As well as the numerical parameters defining the potentials, DL_POLY_2 must also be provided with a cutoff radius r_{cut}, which sets a range limit on the computation of the interaction. Together with the parameters, the cutoff is used by the subroutine FORGEN (or FORGEN_RSQ) to construct an interpolation array `vvv` for the potential function over the range 0 to r_{cut}. A second array `ggg` is also calculated, which is related to the potential via the formula:

$$G(r_{ij}) = -r_{ij}\frac{\partial}{\partial r_{ij}}U(r_{ij}), \tag{2.86}$$

and is used in the calculation of the forces. Both arrays are tabulated in units of energy. The use of interpolation arrays, rather than the explicit formulae, makes the routines for

calculating the potential energy and atomic forces very general, and enables the use of user defined pair potential functions. DL_POLY_2 also allows the user to read in the interpolation arrays directly from a file (see the description of the TABLE file (section 4.1.5). This is particularly useful if the pair potential function has no simple analytical description (e.g. spline potentials).

The force on an atom j derived from one of these potentials is formally calculated with the standard formula:

$$\underline{f}_j = -\frac{1}{r_{ij}}\left[\frac{\partial}{\partial r_{ij}}U(r_{ij})\right]\underline{r}_{ij}, \tag{2.87}$$

where $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$. The force on atom i is the negative of this.

The contribution to be added to the atomic virial (for each pair interaction) is

$$\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j. \tag{2.88}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.89}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived from the pair forces is symmetric.

Since the calculation of pair potentials assumes a spherical cutoff (r_{cut}) it is necessary to apply a *long range correction* to the system potential energy and virial. Explicit formulae are needed for each case and are derived as follows. For two atom types a and b, the correction for the potential energy is calculated via the integral

$$U_{corr}^{ab} = 2\pi\frac{N_a N_b}{V}\int_{r_{cut}}^{\infty} g_{ab}(r)U_{ab}(r)r^2 dr \tag{2.90}$$

where N_a, N_b are the numbers of atoms of types a and b, V is the system volume and $g_{ab}(r)$ and $U_{ab}(r)$ are the appropriate pair correlation function and pair potential respectively. It is usual to assume $g_{ab}(r) = 1$ for $r > r_{cut}$. DL_POLY_2 sometimes makes the additional assumption that the repulsive part of the short ranged potential is negligible beyond r_{cut}.

The correction for the system virial is

$$\mathcal{W}_{corr}^{ab} = -2\pi\frac{N_a N_b}{V}\int_{r_{cut}}^{\infty} g_{ab}(r)\frac{\partial}{\partial r}U_{ab}(r)r^3 dr, \tag{2.91}$$

where the same approximations are applied. Note that these formulae are based on the assumption that the system is reasonably isotropic beyond the cutoff.

In DL_POLY_2 the short ranged forces are calculated by one of the routines SRFRCE, SRFRCE_RSQ, and SRFRCENEU. The long range corrections are calculated by routine LRCORRECT. The calculation makes use of the Verlet neighbour list described above.

2.3.2 Three Body Potentials

The three-body potentials in DL_POLY_2 are mostly valence angle forms. (They are primarily included to permit simulation of amorphous materials e.g. silicate glasses.) However, these have been extended to include the Dreiding [6] hydrogen bond. The potential forms available are as follows.

1. Truncated harmonic: (**thrm**)

$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]; \tag{2.92}$$

2. Screened Harmonic: (**shrm**)

$$U(\theta_{jik}) = \frac{k}{2}(\theta_{jik} - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]; \tag{2.93}$$

3. Screened Vessal[24]: (**bvs1**)

$$\begin{aligned} U(\theta_{jik}) &= \frac{k}{8(\theta_{jik} - \pi)^2} \left\{ \left[(\theta_0 - \pi)^2 - (\theta_{jik} - \pi)^2 \right]^2 \right\} \\ &\quad \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]; \end{aligned} \tag{2.94}$$

4. Truncated Vessal[25]: (**bvs2**)

$$\begin{aligned} U(\theta_{jik}) &= k[\theta_{jik}^a(\theta_{jik} - \theta_0)^2(\theta_{jik} + \theta_0 - 2\pi)^2 - \frac{a}{2}\pi^{a-1} \\ &\quad (\theta_{jik} - \theta_0)^2(\pi - \theta_0)^3] \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]. \end{aligned} \tag{2.95}$$

5. Dreiding hydrogen bond [6]: (**hbnd**

$$U(\theta_{jik}) = D_{hb} cos^4(\theta_{jik})[5(R_{hb}/r_{jk})^{12} - 6(R_{hb}/r_{jk})^{10}] \tag{2.96}$$

Note that for the hydrogen bond, the hydrogen atom *must* be the central atom. Several of these functions are identical to those appearing in the *intra*-molecular valenceangle descriptions above. There are significant differences in implementation however, arising from the fact that the three-body potentials are regarded as *inter*-molecular. Firstly, the atoms involved are defined by atom types, not specific indices. Secondly, there are *no* excluded atoms arising from the three body terms. (The inclusion of pair potentials may in fact be essential to maintain the structure of the system.)

The three body potentials are very short ranged, typically of order 3 $\mathring{A}$. This property, plus the fact that three body potentials scale as N^3, where N is the number of particles, makes it essential that these terms are calculated by the link-cell method [28].

The calculation of the forces, virial and stress tensor as described in the section valence angle potentials above.

DL_POLY_2 applies no long range corrections to the three body potentials. The three body forces are calculated by the routine THBFRC.

2.3.3 The Tersoff Covalent Potential

The Tersoff potential [4] is a special example of a density dependent potential, which has been designed to reproduce the properties of covalent bonding in systems containing carbon, silicon, germanium etc and alloys of these elements. A special feature of the potential is that it allows bond breaking and associated changes in bond hybridisation. The potential has 11 atomic and 2 bi-atomic parameters. The energy is modelled as a sum of pair-like interactions where, however, the coefficient of the attractive term in the pairlike potential (which plays the role of a bond order) depends on the local environment giving a many-body potential.

The form of the Tersoff potential is: (**ters**)

$$U_{ij} = f_C(r_{ij})\ [f_R(r_{ij}) - \gamma_{ij}\ f_A(r_{ij})], \tag{2.97}$$

where

$$f_R(r_{ij}) = A_{ij}\ \exp(-a_{ij}\ r_{ij})\ ,\quad f_A(r_{ij}) = B_{ij}\ \exp(-b_{ij}\ r_{ij}) \tag{2.98}$$

$$f_C(r_{ij}) = \begin{cases} 1 & : \quad r_{ij} < R_{ij} \\ \frac{1}{2} + \frac{1}{2}\cos[\pi\ (r_{ij} - R_{ij})/(r_{ij} - R_{ij})] & : \quad R_{ij} < r_{ij} < S_{ij} \\ 0 & : \quad r_{ij} > S_{ij} \end{cases} \tag{2.99}$$

$$\gamma_{ij} = \chi_{ij}\ (1 + \beta_i^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i})^{-1/2\eta_i}\ ,\quad \mathcal{L}_{ij} = \sum_{k \neq i,j} f_C(r_{ik})\ \omega_{ik}\ g(\theta_{ijk})$$
$$g(\theta_{ijk}) = 1 + c_i^2/d_i^2 - c_i^2/[d_i^2 + (h_i - \cos\theta_{ijk})^2] \tag{2.100}$$

with further mixed parameters defined as

$$\begin{aligned} a_{ij} &= (a_i + a_j)/2 \quad , \quad b_{ij} = (b_i + b_j)/2 \\ A_{ij} &= (A_i A_j)^{1/2} \quad , \quad B_{ij} = (B_i B_j)^{1/2} \\ R_{ij} &= (R_i R_j)^{1/2} \quad , \quad S_{ij} = (S_i S_j)^{1/2} \quad . \end{aligned} \tag{2.101}$$

Here i, j and k label the atoms in the system, r_{ij} is the length of the ij bond, and θ_{ijk} is the bond angle between bonds ij and ik. Single subscripted parameters (11), such as a_i and η_i, depend only on the type of atom.

The chemistry between different atom types is encapsulated in the two sets of bi-atomic parameters χ_{ij} and ω_{ij}:

$$\begin{aligned} \chi_{ii} &= 1 \quad , \quad \chi_{ij} = \chi_{ji} \\ \omega_{ii} &= 1 \quad , \quad \omega_{ij} = \omega_{ji} \quad , \end{aligned} \tag{2.102}$$

which define only one independent parameter for each pair of atom types. The χ parameter is used to strengthen or weaken the heteropolar bonds, relative to the value obtained by simple interpolation. The ω parameter is used to permit greater flexibility when dealing with more drastically different types of atoms.

The force on an atom ℓ derived from this potential is formally calculated with the formula:

$$f_\ell^\alpha = -\frac{\partial}{\partial r_\ell^\alpha} E_{\texttt{tersoff}} = \frac{1}{2}\sum_{i\neq j} -\frac{\partial}{\partial r_\ell^\alpha} U_{ij} \quad , \tag{2.103}$$

with atomic label ℓ being one of i, j, k and α indicating the x, y, z component. The derivative in the above formula expands into

$$-\frac{\partial U_{ij}}{\partial r_\ell^\alpha} = -\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_R(r_{ij}) + \gamma_{ij}\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_A(r_{ij}) + f_C(r_{ij}) f_A(r_{ij})\frac{\partial}{\partial r_\ell^\alpha}\gamma_{ij} \quad , \tag{2.104}$$

with the contributions from the first two terms being:

$$\begin{aligned} -\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_R(r_{ij}) &= -\left\{ f_C(r_{ij})\frac{\partial}{\partial r_{ij}} f_R(r_{ij}) + f_R(r_{ij})\frac{\partial}{\partial r_{ij}} f_C(r_{ij}) \right\} \times \\ &\quad \left\{ \delta_{j\ell}\frac{r_{i\ell}^\alpha}{r_{i\ell}} - \delta_{i\ell}\frac{r_{\ell j}^\alpha}{r_{\ell j}} \right\} \end{aligned} \tag{2.105}$$

$$\begin{aligned} \gamma_{ij}\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{ij}) f_A(r_{ij}) &= \gamma_{ij}\left\{ f_C(r_{ij})\frac{\partial}{\partial r_{ij}} f_A(r_{ij}) + f_A(r_{ij})\frac{\partial}{\partial r_{ij}} f_C(r_{ij}) \right\} \times \\ &\quad \left\{ \delta_{j\ell}\frac{r_{i\ell}^\alpha}{r_{i\ell}} - \delta_{i\ell}\frac{r_{\ell j}^\alpha}{r_{\ell j}} \right\} \quad , \end{aligned} \tag{2.106}$$

and from the third (angular) term:

$$\begin{aligned} f_C(r_{ij}) f_A(r_{ij})\frac{\partial}{\partial r_\ell^\alpha}\gamma_{ij} &= f_C(r_{ij}) f_A(r_{ij})\ \chi_{ij} \quad \times \\ &\quad \left(-\frac{1}{2}\right)\left(1+\beta_i{}^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i}\right)^{-\frac{1}{2\eta_i}-1}\beta_i{}^{\eta_i}\ \mathcal{L}_{ij}^{\eta_i-1}\frac{\partial}{\partial r_\ell^\alpha}\mathcal{L}_{ij} \quad , \end{aligned} \tag{2.107}$$

where

$$\frac{\partial}{\partial r_\ell^\alpha}\mathcal{L}_{ij} = \frac{\partial}{\partial r_\ell^\alpha}\sum_{k\neq i,j}\omega_{ik}\ f_C(r_{ik})\ g(\theta_{ijk}) \quad . \tag{2.108}$$

The angular term can have three different contributions depending on the index of the particle participating in the interaction:

$$\ell = i \quad : \quad \frac{\partial}{\partial r_i^\alpha}\mathcal{L}_{ij} = \sum_{k\neq i,j}\omega_{ik}\left[g(\theta_{ijk})\frac{\partial}{\partial r_i^\alpha} f_C(r_{ik}) + f_C(r_{ik})\frac{\partial}{\partial r_i^\alpha} g(\theta_{ijk}) \right] \tag{2.109}$$

$$\ell = j \quad : \quad \frac{\partial}{\partial r_j^\alpha}\mathcal{L}_{ij} = \sum_{k\neq i,j}\omega_{ik}\ f_C(r_{ik})\frac{\partial}{\partial r_j^\alpha} g(\theta_{ijk}) \tag{2.110}$$

$$\ell \neq i,j \quad : \quad \frac{\partial}{\partial r_\ell^\alpha}\mathcal{L}_{ij} = \omega_{i\ell}\left[g(\theta_{ij\ell})\frac{\partial}{\partial r_\ell^\alpha} f_C(r_{i\ell}) + f_C(r_{i\ell})\frac{\partial}{\partial r_\ell^\alpha} g(\theta_{ij\ell}) \right] \quad . \tag{2.111}$$

The derivative of $g(\theta_{ijk})$ is worked out in the following manner:

$$\frac{\partial}{\partial r_\ell^\alpha} g(\theta_{ijk}) = \frac{\partial g(\theta_{ijk})}{\partial \theta_{ijk}} \; \frac{-1}{\sin\theta_{ijk}} \; \frac{\partial}{\partial r_\ell^\alpha} \left\{ \frac{\underline{r}_{ij} \cdot \underline{r}_{ik}}{r_{ij} \; r_{ik}} \right\} \quad , \tag{2.112}$$

where

$$\frac{\partial g(\theta_{ijk})}{\partial \theta_{ijk}} = \frac{2 \; c_i^2 (h_i - \cos\theta_{ijk}) \; \sin\theta_{ijk}}{[d_i^2 + (h_i - \cos\theta_{ijk})^2]^2} \tag{2.113}$$

$$\begin{aligned} \frac{\partial}{\partial r_\ell^\alpha} \left\{ \frac{\underline{r}_{ij} \cdot \underline{r}_{ik}}{r_{ij} r_{ik}} \right\} &= (\delta_{\ell j} - \delta_{\ell i}) \frac{r_{ik}^\alpha}{r_{ij} r_{ik}} + (\delta_{\ell k} - \delta_{\ell i}) \frac{r_{ij}^\alpha}{r_{ij} r_{ik}} - \\ & \cos(\theta_{jik}) \left\{ (\delta_{\ell j} - \delta_{\ell i}) \frac{r_{ij}^\alpha}{r_{ij}^2} + (\delta_{\ell k} - \delta_{\ell i}) \frac{r_{ik}^\alpha}{r_{ik}^2} \right\} \quad . \end{aligned} \tag{2.114}$$

The contribution to be added to the atomic virial can be derived as

$$\mathcal{W} = 3V \frac{\partial E_{\texttt{tersoff}}}{\partial V} = \frac{3 \; V}{2} \sum_{i \neq j} \frac{\partial U_{ij}}{\partial V} \tag{2.115}$$

$$\begin{aligned} \mathcal{W} = \frac{1}{2} \sum_{i \neq j} & \left[\frac{\partial}{\partial r_{ij}} f_C(r_{ij}) f_R(r_{ij}) - \gamma_{ij} \frac{\partial}{\partial r_{ij}} f_C(r_{ij}) f_A(r_{ij}) \right] r_{ij} - \\ & \left(-\frac{1}{2} \right) f_C(r_{ij}) f_A(r_{ij}) \; \chi_{ij} \left(1 + \beta_i^{\eta_i} \; \mathcal{L}_{ij}^{\eta_i} \right)^{-\frac{1}{2\eta_i} - 1} \beta_i^{\eta_i} \; \mathcal{L}_{ij}^{\eta_i - 1} \times \\ & \sum_{k \neq i,j} \omega_{ik} \; g(\theta_{ijk}) \left[\frac{\partial}{\partial r_{ik}} f_C(r_{ik}) \right] r_{ik} \quad . \end{aligned} \tag{2.116}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = -r_i^\alpha f_i^\beta \quad , \tag{2.117}$$

where α and β indicate the x, y, z components. The stress tensor is symmetric.

Interpolation arrays, `vmbp` and `gmbp` (set up in subroutine TERGEN) - similar to those in van der Waals interactions 2.3.1 - are used in the calculation of the Tersoff forces, virial and stress.

The Tersoff potentials are very short ranged, typically of order 3 Å. This property, plus the fact that Tersoff potentials (two- and three-body contributions) scale as N^3, where N is the number of particles, makes it essential that these terms are calculated by the link-cell method [28].

DL_POLY_2 applies no long range corrections to the Tersoff potentials. In DL_POLY_2 Tersoff forces are handled by the routines TERSOFF, TERINT and TERSOFF3.

2.3.4 Four Body Potentials

The four-body potentials in DL_POLY_2 are entirely inversion angle forms, primarily included to permit simulation of amorphous materials (particularly borate glasses). The potential forms available in DL_POLY_2 are as follows.

1. Harmonic: (**harm**)

$$U(\phi_{ijkn}) = \frac{1}{2}k(\phi_{ijkn} - \phi_0)^2 \tag{2.118}$$

2. Harmonic cosine: (**hcos**)

$$U(\phi_{ijkn}) = \frac{k}{2}(cos(\phi_{ijkn}) - cos(\phi_0))^2 \tag{2.119}$$

3. Planar potential: (**plan**)

$$U(\phi_{ijkn}) = A[1 - \cos(\phi_{ijkn})] \tag{2.120}$$

These functions are identical to those appearing in the *intra*-molecular inversion angle descriptions above. There are significant differences in implementation however, arising from the fact that the four-body potentials are regarded as *inter*-molecular. Firstly, the atoms involved are defined by atom types, not specific indices. Secondly, there are *no* excluded atoms arising from the four-body terms. (The inclusion of other potentials, for example pair potentials, may in fact be essential to maintain the structure of the system.)

The four body potentials are very short ranged, typically of order 3 $\mathring{A}$. This property, plus the fact that four body potentials scale as N^4, where N is the number of particles, makes it essential that these terms are calculated by the link-cell method [28].

The calculation of the forces, virial and stress tensor described in the section on inversion angle potentials above.

DL_POLY_2 applies no long range corrections to the four body potentials. The four-body forces are calculated by the routine FBPFRC.

2.3.5 Metal Potentials

The metal potentials in DL_POLY_2 follow two similar but distinct formalisms. The first of these is the embedded atom model (EAM) [29, 30] and the second is the Finnis-Sinclair model (FSM) [31]. Both are density dependent potentials derived from density functional theory (DFT) and describe the bonding of a metal atom ultimately in terms of the local electronic density. They are suitable for calculating the properties of metals and metal alloys.

For single component metals the two approaches are the same. **However** they are subtly different in the way they are extended to handle alloys (see below). It follows that EAM and FSM potentials cannot be mixed in a single simulation. Furthermore, even for FSM potentials possessing different analytical forms there is no agreed procedure for mixing the parameters. The user is therefore strongly advised to be consistent in the choice of potential when modelling alloys.

The general form of the EAM and FSM potentials is [32]

$$U_{metal} = \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} V_{ij}(r_{ij}) + \sum_{i=1}^{N} F(\rho_i) \quad , \tag{2.121}$$

where $F(\rho_i)$ is a functional describing the energy of embedding an atom in the bulk density, ρ_i, which is defined as

$$\rho_i = \sum_{j=1,j\neq i}^{N} \rho_{ij}(r_{ij}) \quad . \tag{2.122}$$

It should be noted that the density is determined by the coordination number of the atom defined by *pairs* of atoms. This makes the metal potential dependent on the local density (environmental). $V_{ij}(r_{ij})$ is a pair potential incorporating repulsive electrostatic and overlap interactions. N is the number of interacting particles in the MD box.

The types of metal potentials available in DL_POLY_2 are as follows:

1. EAM potential: (**eam**) There are no explicit mathematical expressions for EAM potentials, so this potential type is read exclusively in the form of interpolation arrays from the TABEAM table file (as implemented in the METTAB routine - Section 4.1.6.) The rules for combining the potentials from different metals to handle alloys are different from the FSM class of potentials (see below).

2. Finnis-Sinclair potential [31]: (**fnsc**) The Finnis-Sinclair potential is explicitly analytical. It has the following form:

$$\begin{aligned} V_{ij}(r_{ij}) &= (r_{ij}-c)^2(c_0+c_1 r_{ij}+c_2 r_{ij}^2) \\ \rho_{ij}(r_{ij}) &= (r_{ij}-c)^2+\beta\frac{(r_{ij}-d)^3}{d} \\ F(\rho_i) &= -A\sqrt{\rho_i} \quad , \end{aligned} \tag{2.123}$$

   with parameters: c_0, c_1, c_2, c, A, d, β, both c and d are cutoffs. Since first being proposed a number of alternative analysical forms have been proposed, some of which are descibed below. The rules for combining different metal potentials to model alloys are different from the EAM potentials (see below).

3. Sutton-Chen potential [3, 33, 34]: (**stch**) The Sutton Chen potential is an analytical potential in the FSM class. It has the form:

$$\begin{aligned} V_{ij}(r_{ij}) &= \epsilon\left(\frac{a}{r_{ij}}\right)^n \\ \rho_{ij}(r_{ij}) &= \left(\frac{a}{r_{ij}}\right)^m \\ F(\rho_i) &= -c\epsilon\sqrt{\rho_i} \quad , \end{aligned} \tag{2.124}$$

   with parameters: ϵ, a, n, m, c.

4. Gupta potential [35]: (**gupt**) The Gupta potential is another analytical potential in the FSM class. It has the form:

$$V_{ij}(r_{ij}) = A\exp\left(-p\frac{r_{ij}-r_0}{r_0}\right)$$

$$\begin{aligned} \rho_{ij}(r_{ij}) &= \exp\left(-2q_{ij}\frac{r_{ij}-r_0}{r_0}\right) \\ F(\rho_i) &= -B\sqrt{\rho_i} \quad , \end{aligned} \tag{2.125}$$

with parameters: A, r_0, p, B, q_{ij}.

All of these metal potentials can be decomposed into pair contributions and thus fit within the general tabulation scheme of DL_POLY_2 , where they are treated as pair interactions (though note that the metal cutoff, $r_{\rm met}$ has nothing to do with short ranged cutoff, $r_{\rm vdw}$). DL_POLY_2 calculates this potential in two stages: the first calculates the local density, ρ_i, for each atom; and the second calculates the potential energy and forces. Interpolation arrays, `vmet`, `gmet` and `fmet` (METGEN, METTAB) are used in both these stages in the same spirit as in the van der Waals interaction calculations.

The total force $\underline{f}_k^{tot}$ on an atom k derived from this potential is calculated in the standard way:

$$\underline{f}_k^{tot} = -\underline{\nabla}_k U_{metal} \quad . \tag{2.126}$$

We rewrite the EAM/FSM potential, (2.121), as

$$\begin{aligned} U_{metal} &= U_1 + U_2 \\ U_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N} V_{ij}(r_{ij}) \\ U_2 &= \sum_{i=1}^{N} F(\rho_i) \quad , \end{aligned} \tag{2.127}$$

where $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$. The force on atom k is the sum of the derivatives of U_1 and U_2 with respect to $\underline{r}_k$, which is recognisable as a sum of pair forces:

1. EAM force

$$\begin{aligned} -\frac{\partial U_1}{\partial \underline{r}_k} &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial \underline{r}_k} = \sum_{j=1,j\neq k}^{N}\frac{\partial V_{kj}(r_{kj})}{\partial r_{kj}}\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \\ -\frac{\partial U_2}{\partial \underline{r}_k} &= -\sum_{i=1}^{N}\frac{\partial F}{\partial \rho_i}\sum_{j\neq i}^{N}\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial \underline{r}_k} \\ &= -\sum_{i=1,i\neq k}^{N}\frac{\partial F}{\partial \rho_i}\frac{\partial \rho_{ik}(r_{ik})}{\partial r_{ik}}\frac{\partial r_{ik}}{\partial \underline{r}_k} - \sum_{j=1,j\neq k}^{N}\frac{\partial F}{\partial \rho_k}\frac{\partial \rho_{kj}(r_{kj})}{\partial r_{kj}}\frac{\partial r_{kj}}{\partial \underline{r}_k} \\ &= \sum_{j=1,j\neq k}^{N}\left(\frac{\partial F}{\partial \rho_k}+\frac{\partial F}{\partial \rho_j}\right)\frac{\partial \rho_{kj}(r_{kj})}{\partial r_{kj}}\frac{\underline{\underline{r}}_{kj}}{r_{kj}} \quad . \end{aligned} \tag{2.128}$$

In DL_POLY_2 the generation of the force arrays from tabulated data (implemented in the METAL_DERIV routine) is done using a five point interpolation precedure.

2. Finnis-Sinclair force

$$
\begin{aligned}
-\frac{\partial U_1}{\partial \underline{r_k}} &= \sum_{j=1,j\neq k}^{N} \left\{ 2(r_{kj}-c)(c_0+c_1 r_{kj}+c_2 r_{kj}^2) + (r_{kj}-c)^2(c_1+2c_2 r_{kj}) \right\} \frac{\underline{\underline{r_{kj}}}}{r_{kj}} \\
-\frac{\partial U_2}{\partial \underline{r_k}} &= -\sum_{j=1,j\neq k}^{N} \frac{A}{2}\left(\frac{1}{\rho_k}+\frac{1}{\rho_j}\right) \left\{ 2(r_{kj}-d) + 3\beta \frac{(r_{kj}-d)^2}{d} \right\} \frac{\underline{\underline{r_{kj}}}}{r_{kj}} \quad . \qquad (2.129)
\end{aligned}
$$

3. Sutton-Chen force

$$
\begin{aligned}
-\frac{\partial U_1}{\partial \underline{r_k}} &= -\sum_{j=1,j\neq k}^{N} n\epsilon \left(\frac{a}{r_{kj}}\right)^n \frac{\underline{\underline{r_{kj}}}}{r_{kj}} \\
-\frac{\partial U_2}{\partial \underline{r_k}} &= \sum_{j=1,j\neq k}^{N} \frac{mc\epsilon}{2}\left(\frac{1}{\rho_k}+\frac{1}{\rho_j}\right)\left(\frac{a}{r_{kj}}\right)^m \frac{\underline{\underline{r_{kj}}}}{r_{kj}} \quad . \qquad (2.130)
\end{aligned}
$$

4. Gupta force

$$
\begin{aligned}
-\frac{\partial U_1}{\partial \underline{r_k}} &= -\sum_{j=1,j\neq k}^{N} \frac{Ap}{r_0} \exp\left(-p\frac{r_{kj}-r_0}{r_0}\right) \frac{\underline{\underline{r_{kj}}}}{r_{kj}} \\
-\frac{\partial U_2}{\partial \underline{r_k}} &= \sum_{j=1,j\neq k}^{N} \frac{Bq_{kj}}{r_0}\left(\frac{1}{\rho_k}+\frac{1}{\rho_j}\right) \exp\left(-2q_{kj}\frac{r_{kj}-r_0}{r_0}\right) \frac{\underline{\underline{r_{kj}}}}{r_{kj}} \quad . \qquad (2.131)
\end{aligned}
$$

With the metal forces thus defined the contribution to be added to the atomic virial *from each atom pair* is then

$$
\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j \quad , \qquad (2.132)
$$

which equates to:

$$
\begin{aligned}
\Psi &= 3V\frac{\partial U}{\partial V} \\
\Psi &= \frac{3}{2}V\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial V} + 3V\sum_{i=1}^{N}\frac{\partial F(\rho_i)}{\partial \rho_i}\frac{\partial \rho_i}{\partial V} = \Psi_1 + \Psi_2 \\
&\frac{\partial r_{ij}}{\partial V} = \frac{\partial V^{1/3}s_{ij}}{\partial V} = \frac{1}{3}V^{-2/3}s_{ij} = \frac{r_{ij}}{3V} \\
\Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}} r_{ij} \qquad (2.133) \\
&\frac{\partial \rho_i}{\partial V} = \frac{\partial}{\partial V}\sum_{j=1,j\neq i}^{N}\rho_{ij}(r_{ij}) = \sum_{j=1,j\neq i}^{N}\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}}\frac{\partial r_{ij}}{\partial V} = \frac{1}{3V}\sum_{j=1,j\neq i}^{N}\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}} r_{ij} \\
\Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\left(\frac{\partial F(\rho_i)}{\partial \rho_i}+\frac{\partial F(\rho_j)}{\partial \rho_j}\right)\frac{\partial \rho_{ij}(r_{ij})}{\partial r_{ij}} r_{ij} \quad .
\end{aligned}
$$

1. EAM virial
   The same as above.

2. Finnis-Sinclair virial

$$\begin{aligned}\Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\left\{2(r_{ij}-c)(c_0+c_1r_{ij}+c_2r_{ij}^2)+(r_{ij}-c)^2(c_1+2c_2r_{ij})\right\}r_{ij}\\ \Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{A}{2}\left(\frac{1}{\rho_i}+\frac{1}{\rho_j}\right)\left\{2(r_{ij}-d)+3\beta\frac{(r_{ij}-d)^2}{d}\right\}r_{ij}a \quad .\end{aligned} \tag{2.134}$$

3. Sutton-Chen virial

$$\begin{aligned}\Psi_1 &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}n\epsilon\left(\frac{a}{r_{ij}}\right)^n\\ \Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{mc\epsilon}{2}\left(\frac{\partial F(\rho_i)}{\partial\rho_i}+\frac{\partial F(\rho_j)}{\partial\rho_j}\right)\left(\frac{a}{r_{ij}}\right)^m \quad .\end{aligned} \tag{2.135}$$

4. Gupta virial

$$\begin{aligned}\Psi_1 &= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{Ap}{r_0}\exp\left(-p\frac{r_{ij}-r_0}{r_0}\right)r_{ij}\\ \Psi_2 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{N}\frac{Bq_{ij}}{r_0}\left(\frac{1}{\rho_k}+\frac{1}{\rho_j}\right)\exp\left(-2q_{ij}\frac{r_{ij}-r_0}{r_0}\right)r_{ij} \quad .\end{aligned} \tag{2.136}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha}f_j^{\beta} \quad , \tag{2.137}$$

where α and β indicate the x, y, z components. The atomic stress tensor is symmetric.

The long-ranged correction for the DL_POLY_2 metal potential is in two parts. Firstly, by analogy with the short ranged potentials, the correction to the local density is

$$\rho_i = \sum_{j=1,j\neq i}^{\infty}\rho_{ij}(r_{ij})$$

$$\rho_i = \sum_{j=1,j\neq i}^{r_{ij}<r_{\rm met}}\rho_{ij}(r_{ij}) + \sum_{j=1,j\neq i}^{r_{ij}\geq r_{\rm met}}\rho_{ij}(r_{ij}) = \rho_i^o + \delta\rho_i \tag{2.138}$$

$$\delta\rho_i = 4\pi\bar{\rho}\int_{r_{\rm met}}^{\infty}\rho_{ij}(r)dr \quad , \tag{2.139}$$

where ρ_i^o is the uncorrected local density and $\bar{\rho}$ is the *mean particle density*. Evaluating the integral part of the above equation yields:

1. EAM density correction
   No long-ranged corrections apply beyond $r_{\rm met}$.

2. Finnis-Sinclair density correction
   No long-ranged corrections apply beyond cutoffs c and d.

3. Sutton-Chen density correction

$$\delta\rho_i = \frac{4\pi\bar{\rho}a^3}{(m-3)}\left(\frac{a}{r_{\rm met}}\right)^{m-3} \quad . \tag{2.140}$$

4. Gupta density correction

$$\delta\rho_i = \frac{2\pi\bar{\rho}r_0}{q_{ij}}\left[r_{\rm met}^2 + 2r_{\rm met}\left(\frac{r_0}{q_{ij}}\right) + 2\left(\frac{r_0}{q_{ij}}\right)^2\right]\exp\left(-2q_{ij}\frac{r_{\rm met}-r_0}{r_0}\right) \quad . \tag{2.141}$$

The density correction is applied immediately after the local density is calculated. The pair term correction is obtained by analogy with the short ranged potentials and is

$$\begin{aligned}
U_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{\infty} V_{ij}(r_{ij}) \\
U_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}<r_{\rm met}} V_{ij}(r_{ij}) + \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}\geq r_{\rm met}} V_{ij}(r_{ij}) = U_1^o + \delta U_1 \\
\delta U_1 &= 2\pi N\bar{\rho}\int_{r_{\rm met}}^{\infty} V_{ij}(r)r^2dr \\
U_2 &= \sum_{i=1}^{N} F(\rho_i^0 + \delta\rho_i) \\
U_2 &= \sum_{i=1}^{N} F(\rho_i^0) + \sum_{i=1}^{N}\frac{\partial F(\rho_i)_0}{\partial\rho_i}\delta\rho_i) = U_2^0 + \delta U_2 \\
\delta U_2 &= 4\pi\bar{\rho}\sum_{i=1}^{N}\frac{\partial F(\rho_i)_0}{\partial\rho_i}\int_{r_{\rm met}}^{\infty}\rho_{ij}(r)r^2dr \quad .
\end{aligned} \tag{2.142}$$

Note: that $\delta U2$ is not required if ρ_i has already been corrected. Evaluating the integral part of the above equations yields:

1. EAM energy correction
   No long-ranged corrections apply beyond $r_{\rm met}$.

2. Finnis-Sinclair energy correction
   No long-ranged corrections apply beyond cutoffs c and d.

3. Sutton-Chen energy correction

$$\begin{aligned}\delta U_1 &= \frac{2\pi N\bar{\rho}\epsilon a^3}{(n-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3} \\ \delta U_2 &= -\frac{4\pi\bar{\rho}a^3}{(m-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3}\left\langle\frac{Nc\epsilon}{2\sqrt{\rho_i^0}}\right\rangle \quad .\end{aligned} \tag{2.143}$$

4. Gupta energy correction

$$\begin{aligned}\delta U_1 &= \frac{2\pi N\bar{\rho}Ar_0}{p}\left[r_{\rm met}^2+2r_{\rm met}\left(\frac{r_0}{p}\right)+2\left(\frac{r_0}{p}\right)^2\right]\times \\ &\quad \exp\left(-p\frac{r_{\rm met}-r_0}{r_0}\right) \\ \delta U_2 &= -\frac{2\pi\bar{\rho}r_0}{q_{ij}}\left[r_{\rm met}^2+2r_{\rm met}\left(\frac{r_0}{q_{ij}}\right)+2\left(\frac{r_0}{q_{ij}}\right)^2\right]\times \\ &\quad \exp\left(-2q_{ij}\frac{r_{\rm met}-r_0}{r_0}\right)\left\langle\frac{NB}{2\sqrt{\rho_i^0}}\right\rangle \quad .\end{aligned} \tag{2.144}$$

To estimate the virial correction we assume the corrected local densities are constants (i.e. independent of distance - at least beyond the range $r_{\rm met}$). This allows the virial correction to be computed by the methods used in the short ranged potentials:

$$\begin{aligned}\Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{\infty}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\ \Psi_1 &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}<r_{\rm met}}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}r_{ij}+\frac{1}{2}\sum_{i=1}^{N}\sum_{j\neq i}^{r_{ij}\geq r_{\rm met}}\frac{\partial V_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\ &= \Psi_1^0+\delta\Psi_1 \\ \delta\Psi_1 &= 2\pi N\bar{\rho}\int_{r_{\rm met}}^{\infty}\frac{\partial V_{ij}(r)}{\partial r_{ij}}r^3dr \\ \Psi_2 &= \sum_{i=1}^{N}\frac{\partial F(\rho_i}{\partial\rho_i}\sum_{j\neq i}^{\infty}\frac{\partial\rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\ \Psi_2 &= \sum_{i=1}^{N}\frac{\partial F(\rho_i}{\partial\rho_i}\sum_{j\neq i}^{r_{ij}<r_{\rm met}}\frac{\partial\rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij}+\sum_{i=1}^{N}\frac{\partial F(\rho_i}{\partial\rho_i}\sum_{j\neq i}^{r_{ij}\geq r_{\rm met}}\frac{\partial\rho_{ij}(r_{ij})}{\partial r_{ij}}r_{ij} \\ &= \Psi_2^0+\delta\Psi_2 \\ \delta\Psi_2 &= 4\pi\bar{\rho}\sum_{i=1}^{N}\frac{\partial F(\rho_i)}{\partial\rho_i}\int_{r_{\rm met}}^{\infty}\frac{\partial\rho_{ij}(r)}{\partial r}r^3dr \quad .\end{aligned} \tag{2.145}$$

Evaluating the integral part of the above equations yields:

1. EAM virial correction
   No long-ranged corrections apply beyond $r_{\rm met}$.

2. Finnis-Sinclair virial correction
   No long-ranged corrections apply beyond cutoffs c and d.

3. Sutton-Chen virial correction

$$\begin{aligned}\delta\Psi_1 &= -n\frac{2\pi N\bar{\rho}\epsilon a^3}{(n-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3}\\ \delta\Psi_2 &= m\frac{4\pi\bar{\rho}a^3}{(m-3)}\left(\frac{a}{r_{\rm met}}\right)^{n-3}\left\langle\frac{Nc\epsilon}{2\sqrt{\rho_i^0}}\right\rangle \quad .\end{aligned} \tag{2.146}$$

4. Gupta virial correction

$$\begin{aligned}\delta\Psi_1 &= -\frac{p}{r_0}\frac{2\pi N\bar{\rho}Ar_0}{p}\left[r_{\rm met}^3+3r_{\rm met}^2\left(\frac{r_0}{p}\right)+6r_{\rm met}\left(\frac{r_0}{p}\right)^2+6\left(\frac{r_0}{p}\right)^3\right]\times\\ &\quad \exp\left(-p\frac{r_{\rm met}-r_0}{r_0}\right)\\ \delta\Psi_2 &= \frac{q_{ij}}{r_0}\frac{2\pi\bar{\rho}r_0}{q_{ij}}\left[r_{\rm met}^3+3r_{\rm met}^2\left(\frac{r_0}{q_{ij}}\right)+6r_{\rm met}\left(\frac{r_0}{q_{ij}}\right)^2+6\left(\frac{r_0}{q_{ij}}\right)^3\right]\times\\ &\quad \exp\left(-2q_{ij}\frac{r_{\rm met}-r_0}{r_0}\right)\left\langle\frac{NB}{2\sqrt{\rho_i^0}}\right\rangle \quad .\end{aligned} \tag{2.147}$$

In the energy and virial corrections we have used the approximation:

$$\sum_i^N \rho_i^{-1/2} = \frac{N}{<\rho_i^{1/2}>} \quad , \tag{2.148}$$

where $<\rho_i^{1/2}>$ is regarded as a constant of the system.

In DL_POLY_2 the metal forces are handled by the routine METFRC. The local density is calculated by the routines METDENS, EAMDEN and FSDEN. The long-ranged corrections are calculated by LRCMETAL. Reading and generation of EAM table data from TABEAM is handled by METTAB and METAL_DERIV.

Notes on the Treatment of Alloys

The distinction to be made between EAM and FSM potentials with regard to alloys concerns the mixing rules for unlike interactions. Starting with equations (2.121) and (2.122), it is clear that we require mixing rules for terms $V_{ij}(r_{ij})$ and $\rho_{ij}(r_{ij})$ when atoms i and j are of different kinds. Thus two different metals A and B we can distinguish 4 possible variants of each:

$$V_{ij}^{AA}(r_{ij}),\ V_{ij}^{BB}(r_{ij}),\ V_{ij}^{AB}(r_{ij}),\ V_{ij}^{BA}(r_{ij})$$

and

$$\rho_{ij}^{AA}(r_{ij}),\ \rho_{ij}^{BB}(r_{ij}),\ \rho_{ij}^{AB}(r_{ij}),\ \rho_{ij}^{BA}(r_{ij}).$$

These forms recognise that the contribution of a type A atom to the potential of a type B atom may be different from the contribution of a type B atom to the potential of a type A atom. In both EAM [36] and FSM [33] cases it turns out that

$$V_{ij}^{BA}(r_{ij}) = V_{ij}^{BA}(r_{ij}) \quad , \tag{2.149}$$

though the mixing rules are different in each case (**beware!**).

With regard to density, in the EAM case it is required that [36]:

$$\begin{aligned} \rho_{ij}^{AB}(r_{ij}) &= \rho_{ij}^{BB}(r_{ij}) \\ \rho_{ij}^{BA}(r_{ij}) &= \rho_{ij}^{AA}(r_{ij}) \quad , \end{aligned} \tag{2.150}$$

which means that an atom of type A contributes the same density to the environment of an atom of type B as it does to an atom of type A, and *vice versa*.

For the FSM case [33] a different rule applies:

$$\rho_{ij}^{AB}(r_{ij}) = (\rho_{ij}^{AA}(r_{ij})\rho_{ij}^{BB}(r_{ij}))^{1/2} \tag{2.151}$$

so that atoms of type A and B contribute the same densities to each other, but not to atoms of the same type.

Thus when specifying these potentials in the DL_POLY_2 FIELD file for an alloy composed of n different metal atom types both EAM and FSM require the specification of $n(n+1)/2$ pair functions $V_{ij}^{AB}(r_{ij})$. However, the EAM requires only n density functions $\rho_{ij}^{AA}(r_{ij})$, whereas the FSM class requires all the cross functions $\rho_{ij}^{AB}(r_{ij})$ or $n(n+1)/2$ in total. In addition to the $n(n+1)/2$ pair functions and n density functions the EAM requires further specification of n functional forms of the density dependence (i.e. the embedding function $F(\rho_i)$ in (2.121)).

For EAM potentials all the functions are supplied in tabular form via the table file TABEAM (see section 4.1.6) to which DL_POLY_2 is redirected by the FIELD file data. The FSM potentials are defined via the necessary parameters in the FIELD file.

2.3.6 External Fields

In addition to the molecular force field, DL_POLY_2 allows the use of an *external* force field. Examples of field available include:

1. Electric field: (**elec**)
$$\underline{F_i} = \underline{F_i} + q_i.\underline{H} \tag{2.152}$$
2. Oscillating shear: (**oshm**)
$$\underline{F}_x = A\cos(2n\pi.z/L_z) \tag{2.153}$$

3. Continuous shear: (**shrx**)

$$\underline{v}_x = \frac{1}{2} A \frac{|z|}{z} \qquad : |z| > z_0 \tag{2.154}$$

4. Gravitational field: (**grav**)

$$\underline{F_i} = \underline{F_i} + m_i . \underline{H} \tag{2.155}$$

5. Magnetic field: (**magn**)

$$\underline{F_i} = \underline{F_i} + q_i . (\underline{v_i} \wedge \underline{H}) \tag{2.156}$$

6. Containing sphere: (**sphr**)

$$\underline{F} = A(R_0 - r)^{-n} \qquad : r > R_{cut} \tag{2.157}$$

7. Repulsive wall: (**zbnd**)

$$\underline{F} = A(z_o - z) \qquad : z > z_o \tag{2.158}$$

It is recommended that the use of an external field should be accompanied by a thermostat (this does not apply to examples 6 and 7, since these are conservative fields). The user is advised to be careful with units!

In DL_POLY_2 external field forces are handled by the routine EXTNFLD.

2.4 Long Ranged Electrostatic (Coulombic) Potentials

DL_POLY_2 incorporates several techniques for dealing with long ranged electrostatic potentials. [2] These are as follows.

1. Atomistic and charge group implementation.
2. Direct Coulomb sum;
3. Truncated and shifted Coulomb sum;
4. Coulomb sum with distance dependent dielectric;
5. Ewald sum;
6. Smoothed Particle Mesh Ewald (SPME);
7. Hautman Klein Ewald for systems with 2D periodicity;
8. Reaction field;
9. Dynamical shell model;

[2] Unlike the other elements of the force field, the electrostatic forces are NOT specified in the input FIELD file, but by setting appropriate directives in the CONTROL file. See section 4.1.1.

10. Relaxed shell model.

Some of these techniques can be combined. For example 1, 3 and 4 can be used in conjunction with 9. The Ewald sum, SPME and Hautman Klein Ewald are restricted to periodic (or pseudo-periodic) systems only, though DL_POLY_2 can handle a broad selection of periodic boundary conditions, including cubic, orthorhombic, parallelepiped, truncated octahedral, hexagonal prism and rhombic dodecahedral. The Ewald sum is the method of choice for periodic systems. The other techniques can be used with either periodic or non-periodic systems, though in the case of the direct Coulomb sum, there are likely to be problems with convergence.

DL_POLY_2 will correctly handle the electrostatics of both molecular and atomic species. However it is assumed that the system is electrically neutral. A warning message is printed if the system is found to be charged, but otherwise the simulation proceeds as normal. No correction for non-neutrality is applied.

2.4.1 Atomistic and Charge Group Implementation

The Ewald sum is an accurate method for summing long-ranged Coulomb potentials in periodic systems. This can be a very cpu intensive calculation and the use of more efficient, but less accurate methods, is common. Invariably this involves truncation of the potential at some finite distance $r_{\rm cut}$. If an atomistic scheme is used for the truncation criterion there is no guarantee that the interaction sphere will be neutral and spurious "charging" effects will almost certainly be seen in a simulation. This arises because the potential being truncated is long-ranged ($1/r$ for charge-charge interactions). However if the cutoff scheme is based on *neutral* groups of atoms, then at worst, at long distance the interaction will be a dipole-dipole interaction and vary as $1/r^3$. The truncation effects at the cutoff are therefore much less severe than if an atomistic scheme is used. In DL_POLY_2 the interaction is evaluated between all atoms of both groups if any site of the first group is within the cutoff distance of any site of the second group. The groups are known interchangeably as "charge groups" or "neutral groups" in the documentation - which serves as a reminder that the advantages of using such a scheme are lost if the groups carry an overall charge. There is no formal requirement in DL_POLY_2 that the groups actually be electrically neutral.

The charge group scheme is more cpu intensive than a simple atomistic cutoff scheme as more computation is required to determine whether or not to include a set of interactions. However the size of the Verlet neighbourhood list (easily the largest array in DL_POLY_2) is considerably smaller with a charge group scheme than an atomistic scheme as only a list of interacting groups need be stored as opposed to a list of interacting atoms.

2.4.2 Direct Coulomb Sum

Use of the direct Coulomb sum is sometimes necessary for accurate simulation of isolated (nonperiodic) systems. It is *not* recommended for periodic systems.

The interaction potential for two charged ions is

$$U(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}} \tag{2.159}$$

with q_ℓ the charge on an atom labelled ℓ, and r_{ij} the magnitude of the separation vector $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$.

The force on an atom j derived from this force is

$$\underline{f}_j = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}^3}\underline{r}_{ij} \tag{2.160}$$

with the force on atom i the negative of this.

The contribution to the atomic virial is

$$\mathcal{W} = -\frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}} \tag{2.161}$$

which is simply the negative of the potential term.

The contribution to be added to the atomic stress tensor is

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.162}$$

where α, β are x, y, z components. The atomic stress tensor is symmetric.

In DL_POLY_2 these forces are handled by the routines COUL0 and COUL0NEU.

2.4.3 Truncated and Shifted Coulomb Sum

This form of the Coulomb sum has the advantage that it drastically reduces the range of electrostatic interactions, without giving rise to a violent step in the potential energy at the cutoff. Its main use is for preliminary preparation of systems and it is not recommended for realistic models.

The form of the potential function is

$$U(r_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0}\left\{\frac{1}{r_{ij}} - \frac{1}{r_{cut}}\right\} \tag{2.163}$$

with q_ℓ the charge on an atom labelled ℓ, r_{cut} the cutoff radius and r_{ij} the magnitude of the separation vector $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$.

The force on an atom j derived from this potential, within the radius r_{cut}, is

$$\underline{f}_j = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}^3}\underline{r}_{ij} \tag{2.164}$$

with the force on atom i the negative of this.

The contribution to the atomic virial is

$$\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j \tag{2.165}$$

which is *not* the negative of the potential term in this case.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.166}$$

where α, β are x, y, z components. The atomic stress tensor is symmetric.

In DL_POLY_2 these forces are handled by the routine COUL1.

A further refinement of this approach is to truncate the $1/r$ potential at $r_{\rm cut}$ and add a linear term to the potential in order to make both the energy and the force zero at the cutoff. The potential is thus

$$U(r_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0}\left[\frac{1}{r_{ij}} + \frac{r_{ij}}{r_{\rm cut}^2} - \frac{2}{r_{\rm cut}}\right] \tag{2.167}$$

with the force on atom j given by

$$\underline{f}_j = \frac{q_i q_j}{4\pi\epsilon_0}\left[\frac{1}{r_{ij}^2} - \frac{1}{r_{\rm cut}^2}\right]\underline{r}_{ij} \tag{2.168}$$

with the force on atom i the negative of this.

This removes the heating effects that arise from the discontinuity in the forces at the cutoff in the simple truncated and shifted potential. The physics of this potential however are little better. It is only recommended for very crude structure optimizations.

The contribution to the atomic virial is

$$\mathcal{W} = -\underline{r}_{ij} \cdot \underline{f}_j \tag{2.169}$$

which is *not* the negative of the potential term.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.170}$$

where α, β are x, y, z components. The atomic stress tensor is symmetric.

In DL_POLY_2 these forces are handled by the routine COUL4.

2.4.4 Coulomb Sum with Distance Dependent Dielectric

This potential attempts to address the difficulties of applying the direct Coulomb sum, without the brutal truncation of the previous case. It hinges on the assumption that the electrostatic forces are effectively 'screened' in real systems - an effect which is approximated by introducing a dielectic term that increases with distance.

The interatomic potential for two charged ions is

$$U(r_{ij}) = \frac{1}{4\pi\epsilon_0\epsilon(r_{ij})}\frac{q_i q_j}{r_{ij}} \tag{2.171}$$

with q_ℓ the charge on an atom labelled ℓ, and r_{ij} the magnitude of the separation vector $\underline{r}_{ij} = \underline{r}_j - \underline{r}_i$. $\epsilon(r)$ is the distance dependent dielectric function. In DL_POLY_2 it is assumed that this function has the form

$$\epsilon(r) = \epsilon r \tag{2.172}$$

where ϵ is a constant. Inclusion of this term effectively accelerates the rate of convergence of the Coulomb sum.

The force on an atom j derived from this potential is

$$\underline{f}_j = \frac{1}{2\pi\epsilon_0\epsilon}\frac{q_i q_j}{r_{ij}^4}\underline{r}_{ij} \tag{2.173}$$

with the force on atom i the negative of this.

The contribution to the atomic virial is

$$\mathcal{W} = -\underline{r}_{ij}\cdot\underline{f}_j \tag{2.174}$$

which is -2 times the potential term.

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = r_{ij}^{\alpha} f_j^{\beta}, \tag{2.175}$$

where α, β are x, y, z components. The atomic stress tensor is symmetric.

In DL_POLY_2 these forces are handled by the routines COUL2 and COUL2NEU.

2.4.5 Ewald Sum

The Ewald sum [11] is the best technique for calculating electrostatic interactions in a periodic (or pseudo-periodic) system.

The basic model for a neutral periodic system is a system of charged point ions mutually interacting via the Coulomb potential. The Ewald method makes two amendments to this simple model. Firstly each ion is effectively neutralised (at long range) by the superposition of a spherical gaussian cloud of opposite charge centred on the ion. The combined assembly of point ions and gaussian charges becomes the *Real Space* part of the Ewald sum, which is now short ranged and treatable by the methods described above (section 2.1). [3] The second modification is to superimpose a second set of gaussian charges, this time with the same charges as the original point ions and again centred on the point ions (so nullifying the effect of the first set of gaussians). The potential due to these gaussians is obtained from Poisson's equation and is solved as a Fourier series in *Reciprocal Space*. The complete Ewald sum requires an additional correction, known as the self energy correction, which arises from a gaussian acting on its own site, and is constant. Ewald's method therefore replaces a potentially infinite sum in real space by two finite sums: one in real space and one in reciprocal space; and the self energy correction.

For molecular systems, as opposed to systems comprised simply of point ions, additional modifications are necessary to correct for the excluded (intra-molecular) Coulombic interactions. In the real space sum these are simply omitted. In reciprocal space however, the effects of individual gaussian charges cannot easily be extracted, and the correction is made in real space. It amounts to removing terms corresponding to the potential energy of an ion ℓ due to the gaussian charge on a neighbouring charge m (or *vice versa*). This correction

[3] Strictly speaking, the real space sum ranges over all periodic images of the simulation cell, but in the DL_POLY_2 implementation, the parameters are chosen to restrict the sum to the simulation cell and its nearest neighbours i.e. the *minimum images* of the cell contents.

appears as the final term in the full Ewald formula below. The distinction between the error function erf and the more usual complementary error function $erfc$ found in the real space sum, should be noted.

The total electrostatic energy is given by the following formula.

$$\begin{aligned} U_c &= \frac{1}{2V_o\epsilon_0}\sum_{\underline{k}\neq\underline{0}}^{\infty}\frac{\exp(-k^2/4\alpha^2)}{k^2}|\sum_j^N q_j\exp(-i\underline{k}\cdot\underline{r}_j)|^2+\frac{1}{4\pi\epsilon_0}\sum_{n<j}^{N^*}\frac{q_jq_n}{r_{nj}}erfc(\alpha r_{nj}) \\ &\quad -\frac{1}{4\pi\epsilon_0}\sum_{molecules}\sum_{\ell\leq m}^{M^*}q_\ell q_m\left\{\delta_{\ell m}\frac{\alpha}{\sqrt{\pi}}+\frac{erf(\alpha r_{\ell m})}{r_{\ell m}^{1-\delta_{\ell m}}}\right\}, \end{aligned} \tag{2.176}$$

where N is the number of ions in the system and N^* the same number discounting any excluded (intramolecular) interactions. M^* represents the number of excluded atoms in a given molecule and includes the atomic self correction. V_o is the simulation cell volume and $\underline{k}$ is a reciprocal lattice vector defined by

$$\underline{k}=\ell\underline{u}+m\underline{v}+n\underline{w} \tag{2.177}$$

where ℓ,m,n are integers and $\underline{u},\underline{v},\underline{w}$ are the *reciprocal space* basis vectors. Both V_o and $\underline{u},\underline{v},\underline{w}$ are derived from the vectors $(\underline{a},\underline{b},\underline{c})$ defining the simulation cell. Thus

$$V_o=|\underline{a}\cdot\underline{b}\times\underline{c}| \tag{2.178}$$

and

$$\begin{aligned} \underline{u} &= 2\pi\frac{\underline{b}\times\underline{c}}{\underline{a}\cdot\underline{b}\times\underline{c}} \\ \underline{v} &= 2\pi\frac{\underline{c}\times\underline{a}}{\underline{a}\cdot\underline{b}\times\underline{c}} \\ \underline{w} &= 2\pi\frac{\underline{a}\times\underline{b}}{\underline{a}\cdot\underline{b}\times\underline{c}}. \end{aligned} \tag{2.179}$$

With these definitions, the Ewald formula above is applicable to general periodic systems. (A small additional modification is necessary for rhombic dodecahedral and truncated octahedral simulation cells [37].)

In practice the convergence of the Ewald sum is controlled by three variables: the real space cutoff r_{cut}; the convergence parameter α and the largest reciprocal space vector $\underline{k}_{max}$ used in the reciprocal space sum. These are discussed more fully in section 3.3.5. DL_POLY_2 can provide estimates if requested (see CONTROL file description 4.1.1.

The force on an atom j is obtained by differentiation and is

$$\begin{aligned} \underline{f}_j &= -\frac{q_j}{V_o\epsilon_0}\sum_{\underline{k}\neq\underline{0}}^{\infty}i\underline{k}\exp(\underline{k}\cdot\underline{r}_j)\frac{\exp(-k^2/4\alpha^2)}{k^2}\sum_n^N q_n\exp(-i\underline{k}\cdot\underline{r}_n) \\ &\quad +\frac{q_j}{4\pi\epsilon_0}\sum_n^{N^*}\frac{q_n}{r_{nj}^3}\left\{erfc(\alpha r_{nj})+\frac{2\alpha r_{nj}}{\sqrt{\pi}}\exp(-\alpha^2r_{nj}^2)\right\}\underline{r}_{nj} \\ &\quad -\frac{q_j}{4\pi\epsilon_0}\sum_\ell^{M^*}\frac{q_\ell}{r_{\ell j}^3}\left\{erf(\alpha r_{\ell j})-\frac{2\alpha r_{\ell j}}{\sqrt{\pi}}\exp(-\alpha^2r_{\ell j}^2)\right\}\underline{r}_{\ell j} \end{aligned} \tag{2.180}$$

The electrostatic contribution to the system virial can be obtained as the negative of the Coulombic energy. However in DL_POLY_2 this formal equality can be used as a check on the convergence of the Ewald sum. The actual electrostatic virial is obtained during the calculation of the diagonal of the stress tensor.

The electrostatic contribution to the stress tensor is given by

$$\begin{aligned}\underline{\underline{\sigma}} &= \frac{1}{2V_o\epsilon_0}\sum_{\underline{k}\neq\underline{0}}^{\infty}\left\{\underline{\underline{\mathbf{1}}}-2\left(\frac{1}{4\alpha^2}+\frac{1}{k^2}\right)\underline{\underline{\mathbf{K}}}\right\}\frac{\exp(-k^2/4\alpha^2)}{k^2}|\sum_{j}^{N}q_j\exp(-i\underline{k}\cdot\underline{r}_j)|^2 \\ &+\frac{1}{4\pi\epsilon_0}\sum_{j<n}^{N^*}\frac{q_jq_n}{r_{nj}^3}\left\{erfc(\alpha r_{nj})+\frac{2\alpha r_{nj}}{\sqrt{\pi}}\exp(-\alpha^2r_{nj}^2)\right\}\underline{\underline{\mathbf{R_{nj}}}} \qquad (2.181)\\ &-\frac{1}{4\pi\epsilon_0}\sum_{j<\ell}^{M^*}\frac{q_jq_\ell}{r_{\ell j}^3}\left\{erf(\alpha r_{\ell j})-\frac{2\alpha r_{\ell j}}{\sqrt{\pi}}\exp(-\alpha^2r_{\ell j}^2)\right\}\underline{\underline{\mathbf{R_{\ell j}}}},\end{aligned}$$

where matrices $\underline{\underline{\mathbf{K}}}$ and $\underline{\underline{\mathbf{R}}}_{\ell\mathbf{j}}$ are defined as follows.

$$K^{\alpha\beta} = k^\alpha k^\beta \qquad (2.182)$$

$$R_{\ell j}^{\alpha\beta} = r_{\ell j}^\alpha r_{\ell j}^\beta \qquad (2.183)$$

In DL_POLY_2 the full Ewald sum is handled by several routines: EWALD1 and EWALD1A handle the reciprocal space terms; EWALD2, EWALD2_2PT, EWALD2_RSQ and EWALD4, EWALD4_2PT handle the real space terms (with the same Verlet neighbour list routines that are used to calculate the short range forces); and EWALD3 calculates the self interaction corrections. It should be noted that the Ewald potential and force interpolation arrays in DL_POLY_2 are `erc` and `fer` respectively.

2.4.6 Smoothed Particle Mesh Ewald

As its name implies the Smoothed Particle Mesh Ewald (SPME) method is a modification of the standard Ewald method. DL_POLY_2 implements the SPME method of Essmann *et al.* [38]. Formally this method is capable of treating van der Waals forces also, but in DL_POLY_2 it is confined to electrostatic forces only. The main difference from the standard Ewald method is in its treatment of the the reciprocal space terms. By means of an interpolation procedure involving (complex) B-splines, the sum in reciprocal space is represented on a three dimensional rectangular grid. In this form the Fast Fourier Transform (FFT) may be used to perform the primary mathematical operation, which is a 3D convolution. The efficiency of these procedures greatly reduces the cost of the reciprocal space sum when the range of $\underline{k}$ vectors is large. The method (briefly) is as follows (for full details see [38]):

1. Interpolation of the $exp(-i\underline{k}\cdot\underline{r}_j)$ terms (given here for one dimension):

$$exp(2\pi iu_jk/L)\approx b(k)\sum_{\ell=-\infty}^{\infty}M_n(u_j-\ell)exp(2\pi ik\ell/K) \qquad (2.184)$$

in which k is the integer index of the $\underline{k}$ vector in a principal direction, K is the total number of grid points in the same direction and u_j is the fractional coordinate of ion j scaled by a factor K (i.e. $u_j = Ks_j^x$). Note that the definition of the B-splines implies a dependence on the integer K, which limits the formally infinite sum over ℓ. The coefficients $M_n(u)$ are B-splines of order n and the factor $b(k)$ is a constant computable from the formula:

$$b(k) = exp(2\pi i(n-1)k/K) \left[\sum_{\ell=0}^{n-2} M_n(\ell+1) exp(2\pi i k\ell/K) \right]^{-1} \tag{2.185}$$

2. Approximation of the structure factor $S(\underline{k})$:

$$S(\underline{k}) \approx b_1(k_1)b_2(k_2)b_3(k_3)Q^\dagger(k_1,k_2,k_3) \tag{2.186}$$

where $Q^\dagger(k_1,k_2,k_3)$ is the discrete Fourier transform of the *charge array* $Q(\ell_1,\ell_2,\ell_3)$ defined as

$$Q(\ell_1,\ell_2,\ell_3) = \sum_{j=1}^{N} q_j \sum_{n_1,n_2,n_3} M_n(u_{1j}-\ell_1-n_1L_1)M_n(u_{2j}-\ell_2-n_2L_2)M_n(u_{3j}-\ell_3-n_3L_3) \tag{2.187}$$

(in which the sums over $n_{1,2,3}$ etc are required to capture contributions from all relevant periodic cell images, which in practice means the nearest images.)

3. Approximating the reciprocal space energy U_{recip}:

$$U_{recip} = \frac{1}{2V_o\epsilon_0} \sum_{k_1,k_2,k_3} G^\dagger(k_1,k_2,k_3)Q(k_1,k_2,k_3) \tag{2.188}$$

in which $G^\dagger$ is the discrete Fourier transform of the function

$$G(k_1,k_2,k_3) = \frac{\exp(-k^2/4\alpha^2)}{k^2} B(k_1,k_2,k_3)(Q^\dagger(k_1,k_2,k_3))^* \tag{2.189}$$

and where

$$B(k_1,k_2,k_3) = |b_1(k_1)|^2|b_2(k_2)|^2|b_3(k_3)|^2 \tag{2.190}$$

and $(Q^\dagger(k_1,k_2,k_3))^*$ is the complex conjgate of $Q^\dagger(k_1,k_2,k_3)$. The function $G(k_1,k_2,k_3)$ is thus a relatively simple product of the gaussian screening term appearing in the conventional Ewald sum, the function $B(k_1,k_2,k_3)$ and the discrete Fourier transform of $Q(k_1,k_2,k_3)$

4. Calculating the atomic forces, which are given formally by:

$$f_j^\alpha = -\frac{\partial U_{recip}}{\partial r_j^\alpha} = -\frac{1}{V_o\epsilon_0} \sum_{k_1,k_2,k_3} G^\dagger(k_1,k_2,k_3) \frac{\partial Q(k_1,k_2,k_3)}{\partial r_j^\alpha} \tag{2.191}$$

Fortunately, due to the recursive properties of the B-splines, these formulae are easily evaluated.

The virial and the stress tensor are calculated in the same manner as for the conventional Ewald sum.

The DL_POLY_2 subroutines required to calculate the SPME contributions are: BSPGEN, which calculates the B-splines; BSPCOE, which calculates B-spline coefficients; SPL_CEXP, which calculates the FFT and B-spline complex exponentials; EWALD_SPME, which calculates the reciprocal space contributions; SPME_FOR, which calculates the reciprocal space forces; and DLPFFT3, which calculates the 3D complex fast Fourier transform (default code only, Cray, SGI, IBM SP machines have their own FFT routines, selected at compile time and the FFTW public FFT is also an option). These subroutines calculate the reciprocal space components of the Ewald sum only, the real-space calculations are performed by EWALD2, EWALD3 and EWALD 4, as for the normal Ewald sum. In addition there are a few minor utility routines : CPY_RTC copies a real array to a complex array; ELE_PRD is an element-for-element product of two arrays; SCL_CSUM is a scalar sum of elements of a complex array; and SET_BLOCK initialises an array to a preset value (usually zero).

2.4.7 Hautman Klein Ewald (HKE)

The method of Hautman and Klein is an adaptation of the Ewald method for systems which are periodic in two dimensions only [39]. (DL_POLY_2 assumes this periodicity is in the XY plane.)

The HKE method gives the following formula for the electrostatic energy of a system of N (nonbonded) ions that is overall charge neutral[4]:

$$\begin{aligned} U_c \;=\; & \frac{1}{4\epsilon_0 A}\sum_{n=0}^{n_{max}} a_n \sum_{i,j}^{N} q_i q_j z_{ij}^{2n} \sum_{\underline{g}\neq\underline{0}} f_n(g;\alpha) g^{2n-1} exp(i\underline{g}\cdot s_{ij}) + \\ & \frac{1}{8\pi\epsilon_0}\sum_{i\neq j}^{N} q_i q_j \sum_{\underline{L}} \left(\frac{1}{r_{ij,L}} - \sum_{n}^{n_{max}} a_n z_{ij}^{2n} \frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}} \right) + \\ & \frac{1}{8\pi\epsilon_0}\sum_{i}^{N} q_i^2 \sum_{\underline{L}} \frac{(1-h_0(L;\alpha))}{L} - \frac{\alpha}{\epsilon_0 \pi^{3/2}} \sum_{i}^{N} q_i^2 \end{aligned} \tag{2.192}$$

In this formula A is the system area (in the XY plane), $\underline{L}$ is a 2D lattice vector representing the 2D periodicity of the system, s_{ij} is the in-plane (XY) component of the interparticle distance r_{ij} and $\underline{g}$ is a reciprocal lattice vector. Thus

$$\underline{L} = \ell_1 \underline{a} + \ell_2 \underline{b}, \tag{2.193}$$

where ℓ_1, ℓ_2 are integers and vectors $\underline{a}$ and $\underline{b}$ are the lattice basis vectors. The reciprocal lattice vectors are:

$$\underline{g} = n_1 \underline{u} + n_2 \underline{v} \tag{2.194}$$

[4]The reader is warned that for the purpose of compatibility with other DL_POLY_2 Ewald routines we have defined $\alpha = 0.5/\alpha_{HK}$, where α_{HK} is the α parameter defined by Hautman and Klein in [39].

where n_1, n_2 are integers $\underline{u}, \underline{v}$ are reciprocal space vectors (defined in terms of the vectors $\underline{a}$ and $\underline{b}$):

$$\begin{aligned} \underline{u} &= 2\pi(b_y, -b_x)^\dagger/(a_x b_y - a_y b_x) \\ \underline{v} &= 2\pi(-a_y, a_x)^\dagger/(a_x b_y - a_y b_x). \end{aligned} \tag{2.195}$$

The functions $h_n(s;\alpha)$ and $f_n(s;\alpha)$ are the HKE convergence functions, in real and reciprocal space respectively. (C.f. the complementary error and gaussian functions of the original Ewald method.) However they occur to higher orders here, as indicated by the sum over subscript n, which corresponds to terms in a Taylor expansion of r^{-1} in s, the in-plane distance [39]. Usually this sum is truncated at $n_{max} = 1$, but in DL_POLY_2 can go as high as $n_{max} = 3$. In the HKE method the convergence functions are defined as follows:

$$h_n(s;\alpha)/s^{2n+1} = \frac{1}{a_n(2n)!}\nabla^{2n}(h_0(s;\alpha)/s) \tag{2.196}$$

with

$$h_0(s;\alpha) = erf(\alpha s) \tag{2.197}$$

and

$$f_n(g;\alpha) = \frac{1}{a_n(2n)!} f_0(g;\alpha) \tag{2.198}$$

with

$$f_0(g;\alpha) = erfc(g/2\alpha). \tag{2.199}$$

In DL_POLY_2 the $h_n(s;\alpha)/s^{2n+1}$ functions are derived by a recursion algorithm, while the $f_n(g;\alpha)$ functions are obtained by direct evaluation. The coefficients a_n are given by

$$a_n = (-1)^n(2n)!/(2^{2n}(n!)^2). \tag{2.200}$$

As pointed out by Hautman and Klein, the equation (2.192) allows separation of the z_{ij}^{2n} components via the binomial expansion, which greatly simplifies the double sum over atoms in reciprocal space. Thus the reciprocal space part of equation (2.192) becomes

$$U_{recip} = \frac{1}{4\epsilon_0 A}\sum_{n=0}^{n_{max}} a_n \sum_{\underline{g}\neq\underline{0}} f_n(g;\alpha) g^{2n-1} \sum_{p=0}^{2n}(-1)^p C_p^{2n} Z_p(\underline{g}) Z_{2n-p}^*(\underline{g}) \tag{2.201}$$

with C_p^{2n} a binomial coefficient and

$$Z_p(\underline{g}) = \sum_{j=1}^{N} q_j z_j^p exp(i\underline{g}\cdot\underline{s_j}) \tag{2.202}$$

The force on an ion is obtained by the usual differentiation, however in this case the z components have different expressions from the x and y.

$$\begin{aligned} -\frac{\partial U_c}{\partial u_j} &= \frac{1}{4\epsilon_0 A}\sum_{\underline{g}\neq\underline{0}}\sum_{n=0}^{n_{max}} a_n f_n(g;\alpha) g^{2n-1} \sum_{p=0}^{2n}(-1)^p C_p^{2n}\left(Z_p(\underline{g})\frac{\partial Z_{2n-p}^*(\underline{g})}{\partial u_j} + Z_{2n-p}^*(\underline{g})\frac{\partial Z_p(\underline{g})}{\partial u_j}\right) \\ &+ \frac{q_j}{4\pi\epsilon_0}\sum_{n=0}^{n_{max}}\sum_{\underline{L}}{}'\sum_{ij}{}' a_n q_i \frac{\partial}{\partial u_j}\left(z_{ij,L}^{2n}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) \end{aligned} \tag{2.203}$$

where u_j is one of x_j, y_j, z_j and (noting for brevity that x and y derivatives are similar)

$$\begin{aligned}\frac{\partial Z_p(\underline{g})}{\partial x_j} &= ig_x q_j z_j^p exp(i\underline{g}\cdot\underline{s_j}) \\ \frac{\partial Z_p(\underline{g})}{\partial z_j} &= pq_j z_j^{p-1} exp(i\underline{g}\cdot\underline{s_j})\end{aligned} \tag{2.204}$$

and

$$\begin{aligned}\frac{\partial}{\partial x_j}\left(z_{ij,L}^{2n}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) &= s_{ij,L}^x \frac{z_{ij,L}^{2n}}{s_{ij,L}}\frac{\partial}{\partial x_j}\left(\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) \\ \frac{\partial}{\partial z_j}\left(z_{ij,L}^{2n}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}\right) &= 2nz_{ij,L}^{2n-1}\frac{h_n(s_{ij,L};\alpha)}{s_{ij,L}^{2n+1}}.\end{aligned} \tag{2.205}$$

In DL_POLY_2 the partial derivatives of $h_n(s_{ij,L};\alpha)/s_{ij,L}^{2n+1}$ are calculated by a recursion algorithm. Note that when $n = 0$ there is no derivative w.r.t. z.

The virial and stress tensor terms in real space may be calculated directly from the pair forces and interatomic distances in the usual way, and need not be discussed further. The calculation of the reciprocal space contributions (the terms involving the $f_n(g;\alpha)$ functions) are more difficult. Firstly however we note that the reciprocal space contributions to σ_{xz}, σ_{yz} and σ_{zz} may be obtained directly from the force calculations thus:

$$\begin{aligned}\sigma_{xz}^{recip} &= \sum_j z_j f_j^x \\ \sigma_{yz}^{recip} &= \sum_j z_j f_j^y \\ \sigma_{zz}^{recip} &= \sum_j z_j f_j^z\end{aligned} \tag{2.206}$$

which renders the calculation of these components trivial. The remaining components are calculated from

$$\begin{aligned}\sigma_{uv}^{recip} = {} & U_{recip}\delta_{uv} + \frac{1}{4\epsilon_0 A}\sum_{n=0}^{n_{max}} a_n \sum_{\underline{g}\neq\underline{0}} g_u g_v \frac{g^{2n-2}}{a_n(2n)!} \\ & \left(\frac{(2n-1)f_0(g;\alpha)}{g} - \frac{1}{\alpha\sqrt{\pi}}exp(-g^2/4\alpha^2)\right) \\ & \sum_{p=0}^{2n}(-1)^p C_p^{2n} Z_p(\underline{g}) Z_{2n-p}^*(\underline{g})\end{aligned} \tag{2.207}$$

where u, v are one or both of the components x, y. Note that, although it is possible to define these contributions to the stress tensor, it is not possible to calculate a pressure from them unless a finite, arbitrary boundary is imposed on the z direction (which is an assumption applied in DL_POLY_2 , but without implications of periodicity in the z-direction). The x, y components define the surface tension however.

For bonded molecules, as with the standard 3D Ewald sum, it is necessary to extract contributions associated with the excluded atom pairs. In the DL_POLY_2 HKE implementation this amounts to an *a posteriori* subtraction of the corresponding coulomb terms.

In DL_POLY_2 the HKE method is handled by several subroutines: HKGEN constructs the $h_n(s;\alpha)$ convergence functions and their derivatives; HKEWALD1 calculates the reciprocal space terms; HKEWALD2 and HKEWALD3 calculate the real space terms and the bonded atom corrections respectively. HKEWALD4 calculates the primary interactions in the multiple timestep implementation.

2.4.8 Reaction Field

In the reaction field method it is assumed that any given molecule is surrounded by a spherical cavity of finite radius within which the electrostatic interactions are calculated explicitly. Outside the cavity the system is treated as a dielectric continuum. The occurence of any net dipole within the cavity induces a polarisation in the dielectric, which in turn interacts with the given molecule. The model allows the replacement of the infinite Coulomb sum by a finite sum plus the reaction field.

The reaction field model coded into DL_POLY_2 is the implementation of Neumann based on charge-charge interactions [40]. In this model, the total Coulombic potential is given by

$$U_c = \frac{1}{4\pi\epsilon_0}\sum_{j<n} q_j q_n \left[\frac{1}{r_{nj}} + \frac{B_0 r_{nj}^2}{2R_c^3}\right] \tag{2.208}$$

where the second term on the right is the reaction field correction to the explicit sum, with R_c the radius of the cavity. The constant B_0 is defined as

$$B_0 = \frac{2(\epsilon_1 - 1)}{(2\epsilon_1 + 1)}, \tag{2.209}$$

with ϵ_1 the dielectric constant outside the cavity. The effective pair potential is therefore

$$U(r_{nj}) = \frac{1}{4\pi\epsilon_0} q_j q_n \left[\frac{1}{r_{nj}} + \frac{B_0 r_{nj}^2}{2R_c^3}\right]. \tag{2.210}$$

This expression unfortunately leads to large fluctuations in the system Coulombic energy, due to the large 'step' in the function at the cavity boundary. In DL_POLY_2 this is countered by subtracting the value of the potential at the cavity boundary from each pair contribution. The term subtracted is

$$\frac{1}{4\pi\epsilon_0}\frac{q_j q_n}{R_c}\left[1 + \frac{B_0}{2}\right]. \tag{2.211}$$

The effective pair force on an atom j arising from another atom n within the cavity is given by

$$\underline{f}_j = \frac{q_j q_n}{4\pi\epsilon_0}\left[\frac{1}{r_{nj}^3} - \frac{B_0}{R_c^3}\right]\underline{r}_{nj}. \tag{2.212}$$

The contribution of each effective pair interaction to the atomic virial is

$$\mathcal{W} = -\underline{r}_{nj} \cdot \underline{f}_j \tag{2.213}$$

and the contribution to the atomic stress tensor is

$$\sigma^{\alpha\beta} = r^{\alpha}_{nj} f^{\beta}_{j}. \tag{2.214}$$

In DL_POLY_2 the reaction field is handled by the routines COUL3 and COUL3NEU.

2.4.9 Dynamical Shell Model

An atom or ion is polarisable if it develops a dipole moment when placed in an electric field. It is commonly expressed by the equation

$$\underline{\mu} = \alpha \underline{E}, \tag{2.215}$$

where $\underline{\mu}$ is the induced dipole and $\underline{E}$ is the electric field. The constant α is the polarisability.

The dynamical shell model is a method of incorporating polarisability into a molecular dynamics simulation. The method used in DL_POLY_2 is that devised by Fincham *et al* [41] and is known as the adiabatic shell model.

In the *static* shell model a polarisable atom is represented by a massive core and massless shell, connected by a harmonic spring, hereafter called the core-shell unit. The core and shell carry different electric charges, the sum of which equals the charge on the original atom. There is no electrostatic interaction (i.e. self interaction) between the core and shell of the same atom. Non-Coulombic interactions arise from the shell alone. The effect of an electric field is to separate the core and shell, giving rise to a *polarisation* dipole. The condition of static equilibrium gives the polarisability as:

$$\alpha = q_s^2/k \tag{2.216}$$

where q_s is the shell charge and k is the force constant of the harmonic spring.

In the adiabatic method, a fraction of the atomic mass is assigned to the shell to permit a dynamical description. The fraction of mass is chosen to ensure that the natural frequency of vibration ν of the harmonic spring (i.e.

$$\nu = \frac{1}{2\pi}\left[\frac{k}{x(1-x)m}\right]^{1/2}, \tag{2.217}$$

with m the atomic mass,) is well above the frequency of vibration of the whole atom in the bulk system. Dynamically the core-shell unit resembles a diatomic molecule with a harmonic bond, however the high vibrational frequency of the bond prevents effective exchange of kinetic energy between the core-shell unit and the remaining system. Therefore, from an initial condition in which the core-shell units have negligible internal vibrational energy, the units will remain close to this condition throughout the simulation. This is essential if the core shell unit is to maintain a net polarisation. (In practice there is a slow leakage of

kinetic energy into the core-shell units, but this should should not amount to more than a few percent of the total kinetic energy.)

The calculation of the virial and stress tensor in this model is based on that for a diatomic molecule with charged atoms. The electrostatic and short ranged forces are calculated as described above. The forces of the harmonic springs are calculated as described for intramolecular harmonic bonds. The relationship between the kinetic energy and the temperature is different however, as the core-shell unit is permitted only three translational degrees of freedom, and the degrees of freedom corresponding to rotation and vibration of the unit are discounted (the kinetic energy of these is regarded as zero).

In DL_POLY_2 the shell forces are handled by the routine SHLFRC. The kinetic energy is calculated by CORSHL and the routine SHQNCH performs the temperature scaling. The dynamical shell model is used in conjunction with the methods for long range forces described above.

2.4.10 Relaxed Shell Model

The relaxed shell model is based on the same electrostatic principles as the dynamical shell model but in this case the shell is assigned a zero mass. This means the shell cannot be driven dynamically and instead the procedure is first to relax the shell to a condition of zero (or at least negligible) force at the start of the integration of the atomic motion and then integrate the motion of the finite mass core by conventional molecular dynamics. The relaxation of the shells in DL_POLY_2 is accomplished using conjugate gradients. Since each timestep of the algorithm entails a minimisation operation the cost per timestep for this algorithm is considerably more than the adiabatic shell model, however the integration timestep permitted is much larger (as much as a factor 10) so evolution through phase space is not necessarily very different in cost. A description of the method is presented in [42].

2.5 Integration algorithms

2.5.1 The Verlet Algorithms

DL_POLY integration algorithms are based on the Verlet scheme, which is both time reversible and simple [11]. It generates trajectories in the microcanonical (NVE) ensemble in which the total energy (kinetic plus potential energy) is conserved. If this property drifts or fluctuates excessively in the course of a simulation it indicates that the timestep is too large or the potential cutoffs too small (relative r.m.s. fluctuations in the total energy of 10^{-5} are typical with this algorithm).

DL_POLY_2 contains two versions of the Verlet algorithm. The first is the Verlet leapfrog (LF) algorithm and the second is the velocity Verlet (VV).

2.5.1.1 Verlet Leapfrog

The LF algorithm requires values of position ($\underline{r}$) and force ($\underline{f}$) at time t while the velocities ($\underline{v}$) are half a timestep behind. The first step is to advance the velocities to $t + (1/2)\Delta t$ by

integration of the force:

$$\underline{v}(t+\frac{1}{2}\Delta t) \leftarrow \underline{v}(t-\frac{1}{2}\Delta t) + \Delta t\ \frac{\underline{f}(t)}{m} \tag{2.218}$$

where m is the mass of a site and Δt is the timestep.

The positions are then advanced using the new velocities:

$$\underline{r}(t+\Delta t) \leftarrow \underline{r}(t) + \Delta t\ \underline{v}(t+\frac{1}{2}\Delta t) \tag{2.219}$$

Molecular dynamics simulations normally require properties that depend on position and velocity *at the same time* (such as the sum of potential and kinetic energy). In the LF algorithm the velocity at time t is obtained from the average of the velocities half a timestep either side of time t:

$$\underline{v}(t) = \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t) + \underline{v}(t+\frac{1}{2}\Delta t)\right] \tag{2.220}$$

The full selection of LF integration algorithms within DL_POLY_2 is as follows:

NVE_1	Verlet leaprog with SHAKE
NVEQ_1	Rigid units with FIQA and SHAKE
NVEQ_2	Linked rigid units with QSHAKE
NVT_B1	Constant T (Berendsen [19]) with SHAKE
NVT_E1	Constant T (Evans [18]) with SHAKE
NVT_H1	Constant T (Hoover [20]) with SHAKE
NVTQ_B1	Constant T (Berendsen [19]) with FIQA and SHAKE
NVTQ_B2	Constant T (Berendsen [19]) with QSHAKE
NVTQ_H1	Constant T (Hoover [20]) with FIQA and SHAKE
NVTQ_H2	Constant T (Hoover [20]) with QSHAKE
NPT_B1	Constant T,P (Berendsen [19]) with FIQA and SHAKE
NPT_H1	Constant T,P+ (Hoover [20]) with SHAKE
NPTQ_B1	Constant T,P (Berendsen [19]) with FIQA and SHAKE
NPTQ_B2	Constant T,P (Berendsen [19]) with QSHAKE
NPTQ_H1	Constant T,P (Hoover [20]) with FIQA and SHAKE
NPTQ_H2	Constant T,P (Hoover [20]) with QSHAKE
NST_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [19]) with SHAKE
NST_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [20]) with SHAKE
NSTQ_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [19]) with FIQA and SHAKE
NSTQ_B2	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [19]) with QSHAKE
NSTQ_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [20]) with FIQA and SHAKE
NSTQ_H2	Constant T,$\underline{\underline{\sigma}}$ (Hoover [20]) with QSHAKE

In the above table the FIQA algorithm is Fincham's Implicit Quaternion Algorithm [14] and QSHAKE is the DL_POLY_2 algorithm combining rigid bonds and rigid bodies in the same molecule [16].

2.5.1.2 Velocity Verlet

The VV algorithm assumes that positions, velocities and forces are known at each full timestep. The algorithm proceeds in two stages as follows.

In the first stage a half step velocity is calculated:

$$\underline{v}(t+\frac{1}{2}\Delta t) \leftarrow \underline{v}(t) + \frac{1}{2}\Delta t\,\frac{\underline{f}(t)}{m} \tag{2.221}$$

and then the full timestep position is obtained:

$$\underline{r}(t+\Delta t) \leftarrow \underline{r}(t) + \Delta t\,\underline{v}(t+\frac{1}{2}\Delta t) \tag{2.222}$$

In the second stage, using the new positions, the next update of the forces $f(t+\Delta t)$ is obtained, from which the full step velocity is calculated using:

$$\underline{v}(t+\Delta t) \leftarrow \underline{v}(t+\frac{1}{2}\Delta t) + \frac{1}{2}\Delta t\,\frac{\underline{f}(t+\Delta t)}{m} \tag{2.223}$$

Thus at the end of the two stages full synchronisation of the positions, forces and velocities is obtained.

The full selection of VV integration algorithms within DL_POLY_2 is as follows:

NVEVV_1	Velocity Verlet with RATTLE
NVEQVV_1	Rigid units with NOSQUISH and RATTLE
NVEQVV_2	Linked rigid units with QSHAKE
NVTVV_B1	Constant T (Berendsen [19]) with RATTLE
NVTVV_E1	Constant T (Evans [18]) with RATTLE
NVTVV_H1	Constant T (Hoover [20]) with RATTLE
NVTQVV_B1	Constant T (Berendsen [19]) with NOSQUISH and RATTLE
NVTQVV_B2	Constant T (Berendsen [19]) with QSHAKE
NVTQVV_H1	Constant T (Hoover [20]) with NOSQUISH and RATTLE
NVTQVV_H2	Constant T (Hoover [20]) with QSHAKE
NPTVV_B1	Constant T,P (Berendsen [19]) with NOSQUISH and RATTLE
NPTVV_H1	Constant T,P+ (Hoover [20]) with RATTLE
NPTQVV_B1	Constant T,P (Berendsen [19]) with NOSQUISH and RATTLE
NPTQVV_B2	Constant T,P (Berendsen [19]) with QSHAKE
NPTQVV_H1	Constant T,P (Hoover [20]) with NOSQUISH and RATTLE
NPTQVV_H2	Constant T,P (Hoover [20]) with QSHAKE
NSTVV_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [19]) with RATTLE
NSTVV_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [20]) with RATTLE
NSTQVV_B1	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [19]) with NOSQUISH and RATTLE
NSTQVV_B2	Constant T,$\underline{\underline{\sigma}}$ (Berendsen [19]) with QSHAKE
NSTQVV_H1	Constant T,$\underline{\underline{\sigma}}$ (Hoover [20]) with NOSQUISH and RATTLE
NSTQVV_H2	Constant T,$\underline{\underline{\sigma}}$ (Hoover [20]) with QSHAKE

In the above table the NOSQUISH algorithm is the rotational algorithm of Miller *et al* [15] and QSHAKE is the DL_POLY_2 algorithm combining rigid bonds and rigid bodies in the same molecule [16].

2.5.1.3 Temperature and Energy Conservation

For both VV and LF the instantaneous temperature can be obtained from the atomic velocities assuming the system has no net momentum:

$$\mathcal{T} = \frac{\sum_{i=1}^{\mathcal{N}} m_i v_i^2(t)}{k_B f} \tag{2.224}$$

where i labels particles (which can be atoms or rigid molecules), $\mathcal{N}$ the number of particles in the system, k_B Boltzmanns constant and f the number of degrees of freedom in the system ($3\mathcal{N} - 3$ if the system is periodic and without constraints).

The total energy of the system is a conserved quantity

$$\mathcal{H}_{\mathrm{NVE}} = U + KE \tag{2.225}$$

where U is the potential energy of the system and KE the kinetic energy at time t.

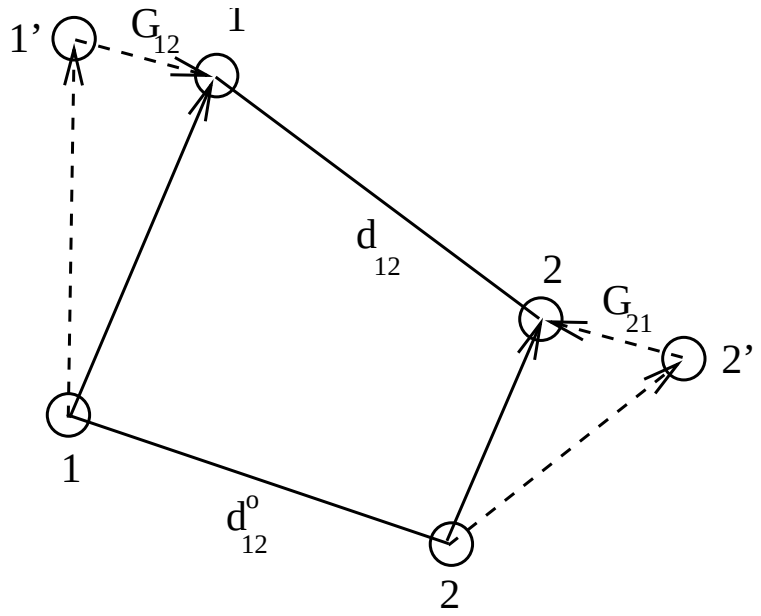
2.5.2 Bond Constraints

2.5.2.1 SHAKE

The SHAKE algorithm for bond constraints was devised by Ryckaert *et al.* [12] and is based on the Verlet leapfrog integration scheme [11]. It is a two stage scheme. In the first stage the leapfrog algorithm calculates the motion of the atoms in the system assuming a complete absence of the rigid bond forces. The positions of the atoms at the end of this stage do not conserve the distance constraint required by the rigid bond and a correction is necessary. In the second stage the deviation in the length of a given rigid bond is used retrospectively to compute the constraint force needed to conserve the bondlength. It is relatively simple to show that the constraint force has the form

$$\underline{G}_{ij} \approx \frac{\mu_{ij}(d_{ij}^2 - d_{ij}'^2)}{2\Delta t^2 \underline{d}_{ij}^o \cdot \underline{d}_{ij}'} \underline{d}_{ij}^o \tag{2.226}$$

where: μ_{ij} is the reduced mass of the two atoms connected by the bond; $\underline{d}_{ij}^o$ and $\underline{d}_{ij}'$ are the original and intermediate bond vectors; d_{ij} is the constrained bondlength; and Δt is the Verlet integration timestep. It should be noted that this formula is an approximation only.



The SHAKE algorithm calculates the constraint force $\underline{G}_{12} = -\underline{G}_{21}$ that conserves the bondlength d_{12} between atoms 1 and 2, following the initial movement to positions $1'$ and $2'$ under the unconstrained forces $\underline{F}_1$ and $\underline{F}_2$.

For a system of simple diatomic molecules, computation of the constraint force will, in principle, allow the correct atomic positions to be calculated in one pass. However in the general polyatomic case this correction is merely an interim adjustment, not only because the above formula is approximate, but the successive correction of other bonds in a molecule has the effect of perturbing previously corrected bonds. The SHAKE algorithm is therfore iterative, with the correction cycle being repeated for all bonds until each has converged to the correct length, within a given tolerance. The tolerance may be of the order 10^{-4} $\mathring{A}$ to 10^{-8} $\mathring{A}$ depending on the precision desired.

The procedure may be summarised as follows:

1. All atoms in the system are moved using the Verlet algorithm, assuming an absence of rigid bonds (constraint forces). (This is stage 1 of the SHAKE algorithm.)

2. The deviation in each bondlength is used to calculate the corresponding constraint force (2.226) that (retrospectively) 'corrects' the bond length.

3. After the correction (2.226) has been applied to all bonds, every bondlength is checked. If the largest deviation found exceeds the desired tolerance, the correction calculation is repeated.

4. Steps 2 and 3 are repeated until all bondlengths satisfy the convergence criterion (This iteration constitutes stage 2 of the SHAKE algorithm).

DL_POLY_2 implements a parallel version of this algorithm [10] (see section 2.6.9). The subroutine NVE_1 implements the Verlet leapfrog algorithm with bond constraints for the NVE ensemble. The routine RDSHAKE_1 is called to apply the SHAKE corrections to position.

It should be noted that the fully converged constraint forces G_{ij} make a contribution to the system virial and the stress tensor.

The contribution to be added to the atomic virial (for each constrained bond) is

$$\mathcal{W} = -\underline{d}_{ij} \cdot \underline{G}_{ij}. \tag{2.227}$$

The contribution to be added to the atomic stress tensor is given by

$$\sigma^{\alpha\beta} = d_{ij}^{\alpha} G_{ij}^{\beta}, \tag{2.228}$$

where α and β indicate the x, y, z components. The atomic stress tensor derived from the pair forces is symmetric.

2.5.2.2 RATTLE

RATTLE [13] is the VV version of SHAKE. It has two parts: the first constrains the bondlength and the second adds an additional constaint to the velocities of the atoms in the constrained bond. The first of these constraints leads to an expression for the constriant force similar to that for SHAKE:

$$\underline{G}_{ij} \approx \frac{\mu_{ij}(d_{ij}^2 - d_{ij}'^2)}{\Delta t^2 \underline{d}_{ij}^o \cdot \underline{d}_{ij}'} \underline{d}_{ij}^o \tag{2.229}$$

Note that this formula differs from equation (2.226) by a factor of 2. This constraint force is applied during the first stage of the velocity Verlet algorithm.

The second constraint condition attempts to maintain the relative velocities of the atoms sharing a bond to a direction perpendicular to the bond vector. This provides another constraint force:

$$\underline{H}_{ij} \approx -\frac{2\mu_{ij}}{\Delta t} \frac{\underline{d}_{ij} \cdot (\underline{v}_j - \underline{v}_i)}{\underline{d}_{ij}^2} \underline{d}_{ij} \tag{2.230}$$

This constraint force is applied during the second stage of the velocity Verlet algorithm. Both constraint force calculations are iterative and are brought to convergence before proceeding to the next stage of the velocity Verlet scheme.

DL_POLY_2 implements a parallel version of RATTLE that is based on the same approach as SHAKE [10] (see section 2.6.9). The subroutine NVEVV_1 implements the velocity Verlet algorithm with bond constraints in the NVE ensemble. The subroutine RDRATTLE_R is called to apply the corrections to atom positions and the subroutine RDRATTLE_V is called to correct the atom velocities.

2.5.3 Potential of Mean Force (PMF) Constraints and the Evaluation of Free Energy

A generalization of bond constraints can be made to constrain a system to some point along a reaction coordinate. A simple example of such a reaction coordinate would be the distance between two ions in solution. If a number of simulations are conducted with the system constrained to different points along the reaction coordinate then the mean constraint force may be plotted as a function of reaction coordinate and the function integrated to obtain the free energy for the overall process [43]. The PMF constraint force, virial and contributions to the stress tensor are obtained in a manner analagous to that for a bond constraint (see previous section). The only difference is that the constraint is now applied between the centres of two groups which need not be atoms alone. DL_POLY_2 reports the PMF constraint virial, W, for each simulation. Users can convert this to the PMF constraint force from

$$G_{\rm PMF} = -\mathcal{W}_{\rm PMF}/d_{\rm PMF}$$

where $d_{\rm PMF}$ is the constraint distance between the two groups used to define the reaction coordinate.

DL_POLY_2 can calculate the PMF using either LF or VV algorithms. Subroutines PMFLF and PMF_SHAKE are used in the LF scheme and subroutines PMFVV, PMF_RATTLE_R and PMF_RATTLE_V are used in the VV scheme.

2.5.4 Thermostats

The system may be coupled to a heat bath to ensure that the average system temperature is maintained close to the requested temperature, $T_{\rm ext}$. When this is done the equations of motion are modified and the system no longer samples the microcanonical ensemble. Instead trajectories in the canonical (NVT) ensemble, or something close to it are generated. DL_POLY_2 comes with three different thermostats: Nosé -Hoover [20], Berendsen [19], and Gaussian constraints [18]. Of these only the Nosé-Hoover algorithm generates trajectories in the canonical (NVT) ensemble. The other methods will produce properties that typically differ from canonical averages by $\mathcal{O}(1/\mathcal{N})$ [11]

2.5.4.1 Nosé - Hoover Thermostat

In the Nosé-Hoover algorithm [20] Newton's equations of motion are modified to read:

$$\begin{aligned} \frac{d\underline{r}(t)}{dt} &= \underline{v}(t) \\ \frac{d\underline{v}(t)}{dt} &= \frac{\underline{f}(t)}{m} - \chi(t)\underline{v}(t) \end{aligned} \tag{2.231}$$

(2.232)

The friction coefficient, χ, is controlled by the first order differential equation

$$\frac{d\chi(t)}{dt} = \frac{N_f k_B}{Q}(\mathcal{T}(t) - T_{\rm ext}) \tag{2.233}$$

where $Q = N_f k_B T_{\rm ext} \tau_T^2$ is the effective 'mass' of the thermoststat, τ_T is a specified time constant (normally in the range [0.5, 2] ps) and N_f is the number of degrees of freedom in the system. $\mathcal{T}(t)$ is the instantaneous temperature of the system at time t.

In the LF version of DL_POLY_2 χ is stored at half timesteps as it has dimensions of (1/time). The integration takes place as:

$$\begin{aligned}
\chi(t+\frac{1}{2}\Delta t) &\leftarrow \chi(t-\frac{1}{2}\Delta t) + \Delta t \frac{N_f k_B}{Q}(\mathcal{T}(t) - T_{\rm ext}) \\
\chi(t) &\leftarrow \frac{1}{2}\left[\chi(t-\frac{1}{2}\Delta t) + \chi(t+\frac{1}{2}\Delta t)\right] \\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}(t-\frac{1}{2}\Delta t) + \Delta t \left[\frac{\underline{f}(t)}{m} - \chi(t)\underline{v}(t)\right] \\
\underline{v}(t) &\leftarrow \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t) + \underline{v}(t+\frac{1}{2}\Delta t)\right] \\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\, \underline{v}(t+\frac{1}{2}\Delta t)
\end{aligned} \tag{2.234}$$

Since $\underline{v}(t)$ is required to calculate $\mathcal{T}(t)$ and itself, the algorithm requires several iterations to obtain self consistency. In DL_POLY_2 the number of iterations is set to 3 (4 if the system has bond constraints). The iteration procedure is started with the standard Verlet leapfrog prediction of $\underline{v}(t)$ and $\mathcal{T}(t)$. The conserved quantity is derived from the extended Hamiltonian for the system which, to within a constant, is the Helmholtz free energy:

$$\mathcal{H}_{\rm NVT} = U + KE + \frac{1}{2}Q\chi(t)^2 + \frac{Q}{\tau_T^2}\int_o^t \chi(s)ds \tag{2.235}$$

If bond constraints are present an extra iteration is required due to the call to the SHAKE routine. The algorithm is implemented in the DL_POLY routine NVT_H1, for systems with bond constraints.

In the VV version of DL_POLY_2 the Hoover algorithm is split into stages in accordance with the principles of Martyna *et al* [17] for designing reversible integrators. The scheme applied here is:

$$\begin{aligned}
\chi(t+\frac{1}{2}\Delta t) &\leftarrow \chi(t) + \frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t) - T_{\rm ext}) \\
\underline{v}'(t) &\leftarrow \underline{v}(t) - \frac{\Delta t}{2}\chi(t+\frac{1}{2}\Delta t)\underline{v}(t) \\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}'(t) + \frac{\Delta t}{2}\frac{\underline{f}(t)}{m} \\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\, \underline{v}(t+\frac{1}{2}\Delta t) \\
& \quad call \quad rattle(R) \\
\underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t) + \frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m}
\end{aligned}$$

$$\begin{aligned} & call \quad rattle(V) \\ \chi(t+\Delta t) \quad & \leftarrow \quad \chi(t+\frac{1}{2}\Delta t) + \frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t+\Delta t) - T_{\rm ext}) \\ \underline{v}(t+\Delta t) \quad & \leftarrow \quad \underline{v}'(t+\Delta t) - \frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}'(t+\Delta t) \end{aligned} \tag{2.236}$$

Routines $rattle(R)$ and $rattle(V)$ apply the bondlength and velocity constraint formulae (2.229) and (2.230) respectively. The equations have the same conserved variable ($\mathcal{H}_{\rm NVT}$) as the LF scheme. The integration is performed by the subroutine NVTVV_H1 which calls subroutines RATTLE_R, RATTLE_V and NVTSCALE.

2.5.4.2 Berendsen Thermostat

In the Berendsen algorithm the instantaneous temperature is pushed towards the desired temperature by scaling the velocities at each step:

$$\chi(t) \quad \leftarrow \quad \left[1 + \frac{\Delta t}{\tau_T}\left(\frac{T_{\rm ext}}{\mathcal{T}(t)} - 1\right)\right]^{1/2} \tag{2.237}$$

The DL_POLY_2 LF routines implement this thermostat as follows.

$$\begin{aligned} \underline{v}(t+\frac{1}{2}\Delta t) \quad & \leftarrow \quad \left[\underline{v}(t-\frac{1}{2}\Delta t) + \Delta t\frac{\underline{f}(t)}{m}\right]\chi(t) \\ \underline{v}(t) \quad & \leftarrow \quad \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t) + \underline{v}(t+\frac{1}{2}\Delta t)\right] \\ \underline{r}(t+\Delta t) \quad & \leftarrow \quad \underline{r}(t) + \Delta t\ \underline{v}(t+\frac{1}{2}\Delta t) \end{aligned} \tag{2.238}$$

As with the Nosé-Hoover thermostat iteration is required to obtain self consistency of $\chi(t)$, $\underline{v}(t)$ and $\mathcal{T}(t)$, although it should be noted χ has different roles in the two thermostats. The Berendsen algorithm conserves total momentum but not energy. Here again the presence of constraint bonds requires an additional iteration with one application of SHAKE corrections. The algorithm is implemented in the DL_POLY routine NVT_B1, for systems including bond constraints.

The VV implementation of Berendsen's algorithm proceeds as folows:

$$\begin{aligned} \underline{v}(t+\frac{1}{2}\Delta t) \quad & \leftarrow \quad \underline{v}(t) + \frac{\Delta t}{2}\frac{\underline{f}(t)}{m} \\ \underline{r}(t+\Delta t) \quad & \leftarrow \quad \underline{r}(t) + \Delta t\ \underline{v}(t+\frac{1}{2}\Delta t) \\ & call \quad rattle(R) \\ \underline{v}'(t+\Delta t) \quad & \leftarrow \quad \underline{v}(t+\frac{1}{2}\Delta t) + \frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m} \\ & call \quad rattle(V) \end{aligned}$$

$$\begin{aligned} \chi &\leftarrow \left[1+\frac{\Delta t}{\tau_T}\left(\frac{\mathcal{T}}{T_{\rm ext}}-1\right)\right]^{1/2} \\ \underline{v}(t+\Delta t) &\leftarrow \chi\,\underline{v}'(t+\Delta t) \end{aligned} \tag{2.239}$$

Routines *rattle*(R) and *rattle*(V) apply the bondlength and velocity constraint formulae (2.229) and (2.230) respectively. The integration is performed by the subroutine NVTVV_B1 which calls subroutines RATTLE_R and RATTLE_V.

2.5.5 Gaussian Constraints

Kinetic temperature can be made a constant of the equations of motion by imposing an additional constraint on the system. If one writes the equations of motions as :

$$\frac{d\underline{r}(t)}{dt} = \underline{v}(t) \tag{2.240}$$

$$\frac{d\underline{v}(t)}{dt} = \frac{\underline{f}(t)}{m} - \chi(t)\underline{v}(t) \tag{2.241}$$

with the temperature constraint

$$\frac{d\mathcal{T}}{dt} \propto \frac{d}{dt}\left(\sum_i (m_i v_i)^2\right) \propto \sum_i m_i^2 \underline{v}_i(t)\cdot\underline{f}_i(t) = 0 \tag{2.242}$$

then choosing

$$\chi = \frac{\sum_i m_i \underline{v}_i(t).\underline{f}_i(t)}{\sum_i m_i^2 v_i^2(t)} \tag{2.243}$$

minimizes the "least squares" differences between the Newtonian and constrained trajectories.

Following Brown and Clarke [44] the algorithm is implemented in the LF scheme by calculating $\eta = 1/(1+\chi\Delta t/2)$

$$\begin{aligned} \eta &\leftarrow \sqrt{\frac{T_{\rm ext}}{\mathcal{T}}} \\ \underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow (2\eta-1)\underline{v}(t-\frac{1}{2}\Delta t) + \eta\Delta t\frac{\underline{f}(t)}{m} \\ \underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\,\underline{v}(t+\frac{1}{2}\Delta t) \end{aligned} \tag{2.244}$$

where $\mathcal{T}$ is obtained from standard Verlet leapfrog integration. Only one iteration is needed (two if the system has bond constraints) to constrain the instantaneous temperature to exactly $T_{\rm ext}$ however energy is not conserved by this algorithm. The algorithm is implemented in the DL_POLY routine NVT_E1 for systems with bond constraints.

The VV implementation of Evan's thermostat is as follows

$$
\begin{aligned}
\chi(t) &\leftarrow \sum_i m_i \underline{v}_i(t).\underline{f}_i(t) / \sum_i m_i^2 v_i^2(t) \\
\underline{v}'(t) &\leftarrow \underline{v}(t) - \frac{\Delta t}{2}\chi(t)\underline{v}(t) \\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}'(t) + \frac{\Delta t}{2}\frac{\underline{\underline{f}}(t)}{m} \\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t\, \underline{v}(t+\frac{1}{2}\Delta t) \\
& call \quad rattle(R) \\
\underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t) + \frac{\Delta t}{2}\frac{\underline{\underline{f}}(t+\Delta t)}{m} \\
& call \quad rattle(V) \\
\chi(t+\Delta t) &\leftarrow \sum_i m_i \underline{v}_i'(t+\Delta t).\underline{f}_i(t+\Delta t) / \sum_i m_i^2 v_i'^2(t+\Delta t) \\
\underline{v}(t+\Delta t) &\leftarrow \underline{v}'(t+\Delta t) - \frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}'(t+\Delta t)
\end{aligned}
\tag{2.245}
$$

Routines $rattle(R)$ and $rattle(V)$ apply the bondlength and velocity constraint formulae (2.229) and (2.230) respectively. The integration is performed by the subroutine NVTVV_E1 which calls subroutines RATTLE_R and RATTLE_V.

2.5.6 Barostats

The size and shape of the simulation cell may be dynamically adjusted by coupling the system to a barostat in order to obtain a desired average pressure (P_{ext}) and/or isotropic stress tensor ($\underline{\underline{\sigma}}$). DL_POLY_2 has two such algorithms: a Hoover barostat and the Berendsen barostat. Only the former has a well defined conserved quantity.

2.5.6.1 The Hoover Barostat

DL_POLY_2 uses the Melchionna modification of the Hoover algorithm [45] in which the equations of motion couple a Nosé - Hoover thermostat and a barostat.

Cell size variation

For isotropic fluctuations the equations of motion are:

$$
\begin{aligned}
\frac{d\underline{r}(t)}{dt} &= \underline{v}(t) + \eta(\underline{r}(t) - \underline{R}_0) \\
\frac{d\underline{v}(t)}{dt} &= \frac{\underline{\underline{f}}(t)}{m} - [\chi(t) + \eta(t)]\,\underline{v}(t) \\
\frac{d\chi(t)}{dt} &= \frac{N_f k_B}{Q}(\mathcal{T}(t) - T_{\text{ext}}) + \frac{1}{Q}(W\eta(t)^2 - k_B T_{\text{ext}}) \\
\frac{d\eta(t)}{dt} &= \frac{3}{W}V(t)(\mathcal{P}(t) - P_{\text{ext}}) - \chi(t)\eta(t)
\end{aligned}
$$

$$\frac{dV(t)}{dt} = [3\eta(t)]V(t) \tag{2.246}$$

where $Q = N_f k_B T_{\rm ext} \tau_T^2$ is the effective 'mass' of the thermostat and $W = N_f k_B T_{\rm ext} \tau_P^2$ is the effective 'mass' of the barostat. N_f is the number of degrees of freedom, η is the barostat friction coefficient, R_0 the system centre of mass, τ_T and τ_P are specified time constants for temperature and pressure fluctuations respectively, $\mathcal{P}(t)$ is the instantaneous pressure and V the system volume.

The conserved quantity is, to within a constant, the Gibbs free energy of the system:

$$\mathcal{H}_{NPT} = U + KE + \mathcal{P}_{\rm ext} V(t) + \frac{1}{2} Q \chi(t)^2 + \frac{1}{2} W \eta(t)^2 + \int_o^t (\frac{Q}{\tau_T^2}\chi(s) + k_B \mathcal{T}_{\rm ext}) ds \tag{2.247}$$

The algorithm is readily implemented in the LF scheme as:

$$\begin{aligned}
\chi(t+\tfrac{1}{2}\Delta t) &\leftarrow \chi(t-\tfrac{1}{2}\Delta t) + \frac{\Delta t N_f k_B}{Q}(\mathcal{T}(t) - T_{\rm ext}) + \frac{\Delta t}{Q}(W\eta(t)^2 - k_B T_{\rm ext}) \\
\chi(t) &\leftarrow \frac{1}{2}\left[\chi(t-\tfrac{1}{2}\Delta t) + \chi(t+\tfrac{1}{2}\Delta t)\right] \\
\eta(t+\tfrac{1}{2}\Delta t) &\leftarrow \eta(t-\tfrac{1}{2}\Delta t) + \Delta t \left\{\frac{3V(t)}{W}(\mathcal{P}(t) - P_{\rm ext}) - \chi(t)\eta(t)\right\} \\
\eta(t) &\leftarrow \frac{1}{2}\left[\eta(t-\tfrac{1}{2}\Delta t) + \eta(t+\tfrac{1}{2}\Delta t)\right] \\
\underline{v}(t+\tfrac{1}{2}\Delta t) &\leftarrow \underline{v}(t-\tfrac{1}{2}\Delta t) + \Delta t \left[\frac{\underline{f}(t)}{m} - [\chi(t) + \eta(t)]\,\underline{v}(t)\right] \\
\underline{v}(t) &\leftarrow \frac{1}{2}\left[\underline{v}(t-\tfrac{1}{2}\Delta t) + \underline{v}(t+\tfrac{1}{2}\Delta t)\right] \\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t) + \Delta t \left(\underline{v}(t+\tfrac{1}{2}\Delta t) + \eta(t+\tfrac{1}{2}\Delta t)\left[\underline{r}(t+\tfrac{1}{2}\Delta t) - \underline{R}_0\right]\right) \\
\underline{r}(t+\tfrac{1}{2}\Delta t) &\leftarrow \frac{1}{2}\left[\underline{r}(t) + \underline{r}(t+\Delta t)\right]
\end{aligned} \tag{2.248}$$

Like the LF Nosé-Hoover thermostat, several iterations are required to obtain self consistency. DL_POLY_2 uses 4 iterations (5 if bond constraints are present) with the standard Verlet leapfrog predictions for the initial estimates of $\mathcal{T}(t)$, $\mathcal{P}(t)$, $\underline{v}(t)$ and $\underline{r}(t+\frac{1}{2}\Delta t)$. Note also that the change in box size requires the SHAKE algorithm to be called each iteration with the new cell vectors and volume obtained from:

$$\begin{aligned}
V(t+\Delta t) &\leftarrow V(t)\exp\left[3\Delta t\ \eta(t+\tfrac{1}{2}\Delta t)\right] \\
\underline{\underline{\mathbf{H}}}(t+\Delta t) &\leftarrow \exp\left[\Delta t\ \eta(t+\tfrac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t)
\end{aligned} \tag{2.249}$$

where $\underline{\underline{\mathbf{H}}}$ is the cell matrix whose columns are the three cell vectors $\underline{a}, \underline{b}, \underline{c}$.

The isotropic changes to cell volume are implemented in the DL_POLY LF routine NPT_H1 which allows for systems containing bond constraints.

The implementation in the VV algorithm follows the scheme:

$$
\begin{aligned}
\chi(t+\frac{1}{2}\Delta t) &\leftarrow \chi(t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t)-T_{\rm ext})+\frac{\Delta t}{2Q}(W\eta(t)^2-k_BT_{\rm ext})\\
\underline{v}'(t) &\leftarrow \underline{v}(t)-\frac{\Delta t}{2}\chi(t+\frac{1}{2}\Delta t)\underline{v}(t)\\
\eta(t+\frac{1}{2}\Delta t) &\leftarrow \eta(t)+\frac{\Delta t}{2}\left\{\frac{3V(t)}{W}(\mathcal{P}(t)-P_{\rm ext})-\chi(t)\eta(t)\right\}\\
\underline{v}''(t) &\leftarrow \underline{v}'(t)-\frac{\Delta t}{2}\eta(t+\frac{1}{2}\Delta t)\underline{v}'(t)\\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}''(t)+\frac{\Delta t}{2}\frac{\underline{f}(t)}{m}\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\,\underline{v}(t+\frac{1}{2}\Delta t)\\
&\quad call\quad rattle(R)\\
V(t+\Delta t) &\leftarrow V(t)\exp\left[3\Delta t\,\eta(t+\frac{1}{2}\Delta t)\right]\\
\underline{\underline{\mathbf{H}}}(t+\Delta t) &\leftarrow \exp\left[\Delta t\,\eta(t+\frac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t)\\
\underline{v}'(t+\Delta t) &\leftarrow \underline{v}(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m}\\
&\quad call\quad rattle(V)\\
\eta(t+\Delta t) &\leftarrow \eta(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\left\{\frac{V(t+\Delta t)}{W}(\mathcal{P}(t+\Delta t)-P_{\rm ext})-\chi(t+\Delta t)\eta(t+\Delta t)\right\}\\
\underline{v}''(t+\Delta t) &\leftarrow \underline{v}'(t+\Delta t)-\frac{\Delta t}{2}\eta(t+\Delta t)\underline{v}'(t+\Delta t)\\
\chi(t+\Delta t) &\leftarrow \chi(t+\frac{1}{2}\Delta t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t+\Delta t)-T_{\rm ext})+\frac{\Delta t}{2Q}(W\eta(t+\Delta t)^2-k_BT_{\rm ext})\\
\underline{v}(t+\Delta t) &\leftarrow \underline{v}''(t+\Delta t)+\frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}''(t+\Delta t) \qquad (2.250)
\end{aligned}
$$

Routines $rattle(R)$ and $rattle(V)$ apply the bondlength and velocity constraint formulae (2.229) and (2.230) respectively. The equations have the same conserved variable ($\mathcal{H}_{\rm NPT}$) as the LF scheme. The integration is performed by the subroutine NVTVV_H1 which calls subroutines RATTLE_R, RATTLE_V, NPTSCALE_T and NPTSCALE_P.

Cell size and shape variation

The isotropic algorithms may be extended to allowing the cell shape to vary by defining η as a tensor, $\underline{\underline{\eta}}$.

The LF equations of motion are implemented as:

$$
\chi(t+\frac{1}{2}\Delta t) \leftarrow \chi(t-\frac{1}{2}\Delta t)+\frac{\Delta t N_f k_B}{Q}(\mathcal{T}(t)-T_{\rm ext})+\frac{\Delta t}{Q}(WTr(\underline{\underline{\eta}}(t))^2-9k_BT_{\rm ext})
$$

$$
\begin{aligned}
\chi(t) &\leftarrow \frac{1}{2}\left[\chi(t-\frac{1}{2}\Delta t)+\chi(t+\frac{1}{2}\Delta t)\right]\\
\underline{\underline{\eta}}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{\underline{\eta}}(t-\frac{1}{2}\Delta t)+\frac{\Delta t V(t)}{W}\left(\underline{\underline{\sigma}}(t)-P_{\rm ext}\underline{\underline{\mathbf{1}}}\right)-\chi(t)Tr(\underline{\underline{\eta}}(t))\\
\underline{\underline{\eta}}(t) &\leftarrow \frac{1}{2}\left[\underline{\underline{\eta}}(t-\frac{1}{2}\Delta t)+\underline{\underline{\eta}}(t+\frac{1}{2}\Delta t)\right]\\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}(t-\frac{1}{2}\Delta t)+\Delta t\left[\frac{\underline{f}(t)}{m}-\left[\chi(t)\underline{\underline{\mathbf{1}}}+\underline{\underline{\eta}}(t)\right]\underline{v}(t)\right]\\
\underline{v}(t) &\leftarrow \frac{1}{2}\left[\underline{v}(t-\frac{1}{2}\Delta t)+\underline{v}(t+\frac{1}{2}\Delta t)\right]\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\left(\underline{v}(t+\frac{1}{2}\Delta t)+\underline{\underline{\eta}}(t+\frac{1}{2}\Delta t)\left[\underline{r}(t+\frac{1}{2}\Delta t)-\underline{R}_0\right]\right)\\
\underline{r}(t+\frac{1}{2}\Delta t) &\leftarrow \frac{1}{2}\left[\underline{r}(t)+\underline{r}(t+\Delta t)\right]
\end{aligned}
\tag{2.251}
$$

where $\underline{\underline{\mathbf{1}}}$ is the identity matrix and $\underline{\underline{\sigma}}$ the pressure tensor. The new cell vectors are calculated from

$$
\underline{\underline{\mathbf{H}}}(t+\Delta t) \leftarrow \exp\left[\Delta t\,\underline{\underline{\eta}}(t+\frac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t) \tag{2.252}
$$

DL_POLY_2 uses a power series expansion truncated at the quadratic term to approximate the exponential of the tensorial term. The new volume is found from

$$
V(t+\Delta t) \leftarrow V(t)\exp\left[\Delta t\,Tr(\underline{\underline{\eta}})\right] \tag{2.253}
$$

The conserved quantity is

$$
\mathcal{H}_{NST}=U+KE+\mathcal{P}_{\rm ext}V(t)+\frac{1}{2}Q\chi(t)^2+\frac{1}{2}WTr(\underline{\underline{\underline{\underline{\eta}}}}(\mathbf{t}))^2+\int_o^t(\frac{Q}{\tau_T^2}\chi(s)+9k_B\mathcal{T}_{\rm ext})ds \tag{2.254}
$$

This algorithm is implemented in the routine NST_H1, with bond constraints.

The VV version of this algorithm is implemented as:

$$
\begin{aligned}
\chi(t+\frac{1}{2}\Delta t) &\leftarrow \chi(t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t)-T_{\rm ext})+\frac{\Delta t}{2Q}(WTr(\underline{\underline{\eta}}(t))^2-9k_BT_{\rm ext})\\
\underline{v}'(t) &\leftarrow \underline{v}(t)-\frac{\Delta t}{2}\chi(t+\frac{1}{2}\Delta t)\underline{v}(t)\\
\underline{\underline{\eta}}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{\underline{\eta}}(t)+\frac{\Delta t}{2}\left\{\frac{V(t)}{W}(\underline{\underline{\sigma}}(t)-P_{\rm ext}\underline{\underline{\mathbf{1}}})-\chi(t)Tr(\underline{\underline{\eta}}(t))\right\}\\
\underline{v}''(t) &\leftarrow \underline{v}'(t)-\frac{\Delta t}{2}\underline{\underline{\eta}}(t+\frac{1}{2}\Delta t)\underline{v}'(t)\\
\underline{v}(t+\frac{1}{2}\Delta t) &\leftarrow \underline{v}''(t)+\frac{\Delta t}{2}\frac{\underline{f}(t)}{m}\\
\underline{r}(t+\Delta t) &\leftarrow \underline{r}(t)+\Delta t\,\underline{v}(t+\frac{1}{2}\Delta t)
\end{aligned}
$$

$$
\begin{aligned}
& call && rattle(R) \\
V(t+\Delta t) & \leftarrow && V(t)\exp\left[3\Delta t\ \eta(t+\frac{1}{2}\Delta t)\right] \\
\underline{\underline{\mathbf{H}}}(t+\Delta t) & \leftarrow && \exp\left[\Delta t\ \eta(t+\frac{1}{2}\Delta t)\right]\underline{\underline{\mathbf{H}}}(t) \\
\underline{v}'(t+\Delta t) & \leftarrow && \underline{v}(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\frac{\underline{f}(t+\Delta t)}{m} \\
& call && rattle(V) \\
\underline{\underline{\eta}}(t+\Delta t) & \leftarrow && \underline{\underline{\eta}}(t+\frac{1}{2}\Delta t)+\frac{\Delta t}{2}\left\{\frac{V(t+\Delta t)}{W}(\underline{\underline{\eta}}(t+\Delta t)-P_{\rm ext}\underline{\underline{\mathbf{1}}})-\chi(t+\Delta t)Tr(\underline{\underline{\eta}}(t+\Delta t))\right\} \\
\underline{v}''(t+\Delta t) & \leftarrow && \underline{v}'(t+\Delta t)-\frac{\Delta t}{2}\underline{\underline{\eta}}(t+\Delta t)\underline{v}'(t+\Delta t) \\
\chi(t+\Delta t) & \leftarrow && \chi(t+\frac{1}{2}\Delta t)+\frac{\Delta t N_f k_B}{2Q}(\mathcal{T}(t+\Delta t)-T_{\rm ext})+\frac{\Delta t}{2Q}(WTr(\underline{\underline{\eta}}(t+\Delta t))^2-9k_BT_{\rm ext}) \\
\underline{v}(t+\Delta t) & \leftarrow && \underline{v}''(t+\Delta t)+\frac{\Delta t}{2}\chi(t+\Delta t)\underline{v}''(t+\Delta t)
\end{aligned}
\tag{2.255}
$$

Routines $rattle(R)$ and $rattle(V)$ apply the bondlength and velocity constraint formulae (2.229) and (2.230) respectively. The equations have the same conserved variable ($\mathcal{H}_{\rm NST}$) as the LF scheme. The integration is performed by the subroutine NVTVV_H1 which calls subroutines RATTLE_R, RATTLE_V, NSTSCALE_T and NSTSCALE_P.

2.5.6.2 Berendsen Barostat

With the Berendsen barostat the system is made to obey the equation of motion

$$\frac{d\mathcal{P}}{dt}=(P_{\rm ext}-\mathcal{P})/\tau_P \tag{2.256}$$

Cell size variations

In the isotropic implementation, at each step the MD cell volume is scaled by by a factor η and the coordinates, and cell vectors, by $\eta^{1/3}$ where

$$\eta=1-\frac{\beta\Delta t}{\tau_P}(P_{\rm ext}-\mathcal{P}) \tag{2.257}$$

and β is the isothermal compressibility of the system. The Berendesen thermostat is applied at the same time. In practice β is a specified constant which DL_POLY_2 takes to be the isothermal compressibility of liquid water. The exact value is not critical to the algorithm as it relies on the ratio τ_P/β. τ_P is specified by the user.

The LF version of this algorithm is implemented in NPT_B1 with 4 or 5 iterations used to obtain self consistency in the $\underline{v}(t)$. It calls RDSHAKE_1 to handle constraints. The VV version is implemented in subroutine NVTVV_B1, which calls constraint subroutines RATTLE_R and RATTLE_V.

Cell size and shape variations

The extension of the isotropic algorithm to anisotropic cell variations is straightforward. The tensor $\underline{\underline{\eta}}$ is defined by

$$\underline{\underline{\eta}} = \underline{\underline{\mathbf{1}}} - \frac{\beta \Delta t}{\tau_P}(P_{\text{ext}}\underline{\underline{\mathbf{1}}} - \underline{\underline{\sigma}}) \tag{2.258}$$

and the new cell vectors given by

$$\underline{\underline{\mathbf{H}}}(t + \Delta t) \leftarrow \underline{\underline{\eta}}\,\underline{\underline{\mathbf{H}}}(t) \tag{2.259}$$

As in the isotropic case the Berendsen thermostat is applied simultaneously and 4 or 5 iterations are used to obtain convergence. The LF version of the algorithm is implemented in subroutine NST_B1 and the VV version in NSTVV_B1. The former calls RDSHAKE_1 to handle constraints and the latter calls subroutines RATTLE_R and RATTLE_V.

2.5.7 Rigid Bodies and Rotational Integration Algorithms

2.5.7.1 Description of Rigid Body Units

A rigid body unit is a collection of point atoms whose local geometry is time invariant. One way to enforce this in a simulation is to impose a sufficient number of bond constraints between the atoms in the unit. However, in many cases this is may be either problematic or impossible. Examples in which it is impossible to specify sufficient bond constraints are

1. linear molecules with more than 2 atoms (e.g. CO_2)
2. planar molecules with more than three atoms (e.g. benzene).

Even when the structure *can* be defined by bond constraints the network of bonds produced may be problematic. Normally, they make the iterative SHAKE procedure slow, particularly if a ring of constraints is involved (as occurs when one defines water as a constrained triangle). It is also possible, inadvertently, to over constrain a molecule (e.g. by defining a methane tetrahedron to have 10 rather than 9 bond constraints) in which case the SHAKE procedure will become unstable. In addition, massless sites (e.g. charge sites) cannot be included in a simple constraint approach making modelling with potentials such as TIP4P water impossible.

All these problems may be circumvented by defining rigid body units, the dynamics of which may be described in terms of the translational motion of the center of mass (COM) and rotation about the COM. To do this we need to define the appropriate variables describing the position, orientation and inertia of a rigid body, and the rigid body equations of motion. [5]

[5] An alternative approach is to define "basic" and "secondary" particles. The basic particles are the minimun number needed to define a local body axis system. The remaining particle positions are expressed in terms of the COM and the basic particles. Ordinary bond constraints can then be applied to the basic particles provided the forces and torques arising from the secondary particles are transferred to the basic particles in a physically meaningful way.

The mass of a rigid unit M is the sum of the atomic masses in that unit:

$$M = \sum_{j=1}^{N_{sites}} m_j. \tag{2.260}$$

where m_j is the mass of an atom and the sum includes all sites (N_{sites}) in the body. The position of the rigid unit is defined as the location of its centre of mass $\underline{R}$:

$$\underline{R} = \frac{1}{M} \sum_{j=1}^{N_{sites}} m_j \underline{r}_j, \tag{2.261}$$

where $\underline{r}_j$ is the position vector of atom j. The rigid body translational velocity $\underline{V}$ is defined by:

$$\underline{V} = \frac{1}{M} \sum_{j=1}^{N_{sites}} m_j \underline{v}_j, \tag{2.262}$$

where $\underline{v}_j$ is the velocity of atom j. The net translational force acting on the rigid body unit is the vector sum of the forces acting on the atoms of the body:

$$\underline{F} = \sum_{j=1}^{N_{sites}} \underline{f}_j \tag{2.263}$$

where $\underline{f}_j$ is the force on a rigid unit site

A rigid body also has associated with it a rotational inertia matrix $\underline{\underline{\mathbf{I}}}$, whose components are given by

$$I_{\alpha\beta} = \sum_{j}^{N_{sites}} m_j (d_j^2 \delta_{\alpha\beta} - d_j^\alpha r_j^\beta) \tag{2.264}$$

where $\underline{d}_j$ is the displacement vector of the atom j from the COM, and is given by:

$$\underline{d}_j = \underline{r}_j - \underline{R}. \tag{2.265}$$

It is common practice in the treatment of rigid body motion to define the position $\underline{R}$ of the body in a universal frame of reference (the so called laboratory or inertial frame), but to describe the moment of inertia tensor in a frame of reference that is localised in the rigid body and changes as the rigid body rotates. Thus the local body frame is taken to be that in which the rotational inertia tensor $\underline{\underline{\hat{\mathbf{I}}}}$ is diagonal and the components satisfy $I_{xx} \geq I_{yy} \geq I_{zz}$. In this local frame (the so called *Principal Frame*) the inertia tensor is therefore constant.

The orientation of the local body frame with respect to the space fixed frame is described via a four dimensional unit vector, the quaternion

$$\underline{q} = [q_0, q_1, q_2, q_3]^T, \tag{2.266}$$

and the rotational matrix $\underline{\underline{\mathbf{R}}}$ to transform from the local body frame to the space fixed frame is the unitary matrix

$$\underline{\underline{\mathbf{R}}} = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2-q_0q_3) & 2(q_1q_3+q_0q_2) \\ 2(q_1q_2+q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3-q_0q_1) \\ 2(q_1q_3-q_0q_2) & 2(q_2q_3+q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix} \tag{2.267}$$

so that if $\underline{\hat{d}}_j$ is the position of an atom in the local body frame (with respect to its COM), its position in the universal frame (w.r.t. its COM) is given by

$$\underline{d}_j = \underline{\underline{\mathbf{R}}}\, \underline{\hat{d}}_j \tag{2.268}$$

With these variables defined we can now consider the equations of motion for the rigid body unit.

2.5.7.2 Integration of the Rigid Body Equations of Motion

The equations of translational motion of a rigid body are the same as those describing the motion of a single atom, except that the force is the total force acting on the rigid body i.e. $\underline{F}$ in equation (2.263) and the mass is the total mass of the rigid body unit i.e. M in equation (2.260). These equations can be integrated by the standard Verlet LF or VV algorithms described in the previous sections. Thus we need only consider the rotational motion here.

The rotational equation of motion for a rigid body is

$$\underline{\tau} = \frac{d}{dt}\underline{J} = \frac{d}{dt}\left(\underline{\underline{\mathbf{I}}}\underline{\omega}\right), \tag{2.269}$$

in which $\underline{J}$ is the angular momentum of the rigid body defined by the expression

$$\underline{J} = \sum_{j=1}^{N_{sites}} m_j \underline{d}_j \times \underline{v}_j, \tag{2.270}$$

and $\underline{\omega}$ is the angular velocity.

The vector $\underline{\tau}$ is the torque acting on the body in the universal frame and is given by

$$\underline{\tau} = \sum_{j=1}^{N_{sites}} \underline{d}_j \times \underline{f}_j. \tag{2.271}$$

The rotational equations of motion, written in the local frame of the rigid body, are given by Euler's equations

$$\begin{aligned} \dot{\hat{\omega}}_x &= \frac{\tau_x}{\hat{I}_{xx}} + (\hat{I}_{yy}-\hat{I}_{zz})\hat{\omega}_y\hat{\omega}_z \\ \dot{\hat{\omega}}_y &= \frac{\tau_y}{\hat{I}_{yy}} + (\hat{I}_{zz}-\hat{I}_{xx})\hat{\omega}_z\hat{\omega}_z \\ \dot{\hat{\omega}}_y &= \frac{\tau_z}{\hat{I}_{zz}} + (\hat{I}_{xx}-\hat{I}_{yy})\hat{\omega}_x\hat{\omega}_y \end{aligned} \tag{2.272}$$

The vector $\underline{\hat{\omega}}$ is the angular velocity transformed to the local body frame. Integration of $\hat{\omega}$ is complicated by the fact that as the rigid body rotates , so does the local reference frame. So is is necessary to integrate equations (2.272) simultaneously with an integration of the quaternions describing the orientiation of the rigid body. The equation describing this is:

$$\begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} 0 \\ \hat{\omega}_x \\ \hat{\omega}_y \\ \hat{\omega}_z \end{pmatrix} \tag{2.273}$$

Rotational motion in DL_POLY_2 is handled by two different methods. For LF implementation, the Fincham Implicit Quaternion Algorithm (FIQA) is used [14]. The VV implementation uses the NOSQUISH algorithm of Miller *et al.* [15].

The LF implementation begins by integrating the angular velocity equation in the local frame.

$$\hat{\omega}(t + \frac{\Delta t}{2}) = \hat{\omega}(t - \frac{\Delta t}{2}) + \Delta t \, \hat{\underline{\underline{\mathbf{I}}}}^{-1} \underline{\hat{\tau}}(t) \tag{2.274}$$

The new quaternions are found using the FIQA algorithm. In this algorithm the new quaternions are found by solving the implicit equation

$$\underline{q}(t + \Delta t) = \underline{q}(t) + \frac{\Delta t}{2} \left(\underline{\underline{\mathbf{Q}}}[\underline{q}(t)] \underline{\hat{w}}(t) + \underline{\underline{\mathbf{Q}}}[\underline{q}(t + \Delta t)] \underline{\hat{w}}(t + \Delta t) \right) \tag{2.275}$$

where $\underline{\hat{w}} = [0, \hat{\omega}]^T$ and $\underline{\underline{\mathbf{Q}}}[\underline{q}]$ is

$$\underline{\underline{\mathbf{Q}}} = \frac{1}{2} \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & -q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \tag{2.276}$$

The above equation is solved iteratively with

$$\underline{q}(t + \Delta t) = \underline{q}(t) + \Delta t \, \underline{\underline{\mathbf{Q}}}[\underline{q}(t)] \underline{\hat{w}}(t) \tag{2.277}$$

as the first guess. Typically no more than 3 or 4 iterations are needed for convergence. At each step the constraint

$$\|\underline{q}(t + \Delta t)\| = 1 \tag{2.278}$$

is imposed.

The NVE LF algorithm is implemented in NVEQ_1 which allows for a system containing a mixture of rigid bodies and atomistic species, provided the rigid bodies are not linked to other species by constraint bonds.

The VV implementation is based on the NOSQUISH algorithm of Miller *et al.* [15]. In addition to the quaternions it requires quaternion momenta defined by

$$\begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix} = 2 \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} 0 \\ \hat{I}_{xx}\hat{\omega}_x \\ \hat{I}_{yy}\hat{\omega}_y \\ \hat{I}_{zz}\hat{\omega}_z \end{pmatrix} \tag{2.279}$$

and quaternion torques defined by

$$\begin{pmatrix} \Upsilon_0 \\ \Upsilon_1 \\ \Upsilon_2 \\ \Upsilon_3 \end{pmatrix} = 2 \begin{pmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{pmatrix} \begin{pmatrix} 0 \\ \hat{\tau}_x \\ \hat{\tau}_y \\ \hat{\tau}_z \end{pmatrix} \tag{2.280}$$

(It should be noted that vectors $\underline{p}$ and $\underline{\Upsilon}$ are 4-component vectors.) The quaternion momenta are first updated a half-step using the formula

$$\underline{p}(t + \frac{\Delta t}{2}) \leftarrow \underline{p}(t) + \frac{\Delta t}{2}\underline{\Upsilon}(t) \tag{2.281}$$

Next a sequence of operations is applied to the quaternions and the quaternion momenta in the order

$$e^{i\mathcal{L}_3(\delta t/2)}e^{i\mathcal{L}_2(\delta t/2)}e^{i\mathcal{L}_1(\delta t)}e^{i\mathcal{L}_2(\delta t/2)}e^{i\mathcal{L}_3(\delta t/2)} \tag{2.282}$$

which preserves the symplecticness of the operations (see reference [17]). Note that δt is some submultiple of Δt. (In DL_POLY_2 the default is $\Delta t = 10\delta t$.) The operators themselves are of the following kind:

$$\begin{aligned} e^{i\mathcal{L}(\delta t)}\underline{q} &= cos(\zeta_k \delta t)\underline{q} + sin(\zeta_k \delta t)P_k\underline{q} \\ e^{i\mathcal{L}(\delta t)}\underline{p} &= cos(\zeta_k \delta t)\underline{p} + sin(\zeta_k \delta t)P_k\underline{p} \end{aligned} \tag{2.283}$$

where P_k is a permutation operator with $k = 0, \ldots, 3$ with the following properties

$$\begin{aligned} P_0\underline{q} &= \{q_0, q_1, q_2, q_3\} \\ P_1\underline{q} &= \{-q_1, q_0, q_3, -q_2\} \\ P_2\underline{q} &= \{-q_2, -q_3, q_0, q_1\} \\ P_3\underline{q} &= \{-q_3, q_2, -q_1, q_0\} \end{aligned} \tag{2.284}$$

and the angular velocity ζ_k is defined as

$$\zeta_k = \frac{1}{4I_k}\underline{p}^T P_k \underline{q}. \tag{2.285}$$

Equations (2.282) to (2.284) represent the heart of the NOSQUISH algorithm and are repeatedly applied (10 times in DL_POLY_2). The final result is the quaternion updated to the full timestep value i.e. $\underline{q}(t + \Delta t)$. These equations form part of the first stage of the VV algorithm.

In the second stage of the VV algorithm, new torques are used to update the quaternion momenta to a full timestep.

$$\underline{p}(t + \Delta t) \leftarrow \underline{p}(t + \frac{\Delta t}{2}) + \frac{\Delta t}{2}\underline{\Upsilon}(t + \Delta t) \tag{2.286}$$

The NVE implementation of this algorithm is in the subroutine NVEQVV_1 which calls the NOSQUISH subroutine to perform the rotation operation. The subroutine also calls RATTLE_R and RATTLE_V to handle any rigid bonds which may be present.

Thermostats and Barostats

It is straightforward to couple the rigid body equations of motion to a thermostat and/or barostat. The thermostat is coupled to both the translational and rotational degrees of freedom and so both the translational and rotational velocities are propagated in an analogous manner to the thermostated atomic velocities. The barostat, however, is coupled only to the translational degrees of freedom and not to the rotation.DL_POLY_2 supports both Hoover and Berendsen thermostats and barostats for systems containing rigid bodies.

For LF integration the Hoover thermostat is implemented in NVTQ_H1, the Hoover isotropic barostat (plus thermostat) in NPTQ_H1 and the anisotropic barostat in NSTQ_H1. The analogous routines for the Berendsen algorithms are NVTQ_B1, NPTQ_B1 and NSTQ_B1. These subroutines also call RDSHAKE_1 to handle any rigid bonds which may be present.

For VV integration the Hoover thermostat is implemented in NVTQVV_H1 (NVTQSCL) , the Hoover isotropic barostat (plus thermostat) in NPTQVV_H1 (NPTQSCL_T, NPTQSCL_P) and the anisotropic barostat in NSTQVV_H1 (NSTQSCL_T, NSTQSCL_P). The analogous routines for the Berendsen algorithms are NVTQVV_B1, NPTQVV_B1 and NSTQVV_B1. (The subroutines in brackets represent supporting subroutines.) These subroutines also call RATTLE_R and RATTLE_V to handle any rigid bonds which may be present.

2.5.7.3 Linked Rigid Bodies

The above integration algorithms can be used for rigid bodies in systems containing "atomic" species (whose equations of motion are integrated with the standard leapfrog algorithm). These rigid bodies may even be linked to other species (including other rigid bodies) by extensible bonds. However if a rigid body is linked to an atom or another rigid body by a bond *constraint* the above algorithms are not adequate. The reason is that the constraint will introduce an additional force and torque on the body that can only be found *after* the integration of the unconstrained unit. DL_POLY_2 has a suite of integration algorithms to cope with this situation in which both the constraint conditions and the quaternion equations are solved similtaneously using an extension of the SHAKE algorithm called "QSHAKE" [16]. It has been cast in both LF and VV forms. We will describe here how it works for VV, the LF version is decribed in [16].

Firstly we assume a rigid body (A) is connected to another (B) at timestep $t_n = n\Delta t$ via bonds between atoms at positions $\underline{r}^n_{Ap}$ and $\underline{r}^n_{Bp}$ given by

$$\begin{aligned} \underline{r}^n_{Ap} &= \underline{R}^n_A + \underline{d}^n_{Ap} \\ \underline{r}^n_{Ap} &= \underline{R}^n_B + \underline{d}^n_{Bp} \end{aligned} \tag{2.287}$$

where $\underline{R}$ represents the rigid body COM and $\underline{d}$ the displacement of the atom from the relevant COM. The subscript p indicates that these are the atoms providing the links.

In the first stage of the VV QSHAKE algorithm, the rigid bodies are allowed to move unrestricted. Our task is then to find the the constraint force $\underline{G}^n_{AB}$ which would preserve the constraint bondlength i.e. $d^n_{ABp} = d^{n+1}_{ABp}$. Assuming we know this force we can write:

$$\underline{R}^{n+1}_A = \underline{\tilde{R}}^{n+1}_A + \frac{\Delta t^2}{2M_A}\underline{G}^n_{AB} \tag{2.288}$$

in which the *tilde* $\tilde{x}$ indicates the corresponding variable computed in the absence of the constraint force. (For brevity, in this and subsequent equations we leave out corresponding equations for body B.)

We can also write the true torque at timestep t_n (i.e. $\underline{\tau}^n$) as

$$\underline{\tau}^n = \underline{\tilde{\tau}}^n + d^n_{Ap} \times \underline{G}^n_{AB}. \tag{2.289}$$

It may be easily shown from this and equation (2.269) that

$$\underline{\dot{\omega}}^n_A = \underline{\tilde{\dot{\omega}}}^n_A + \left(\underline{\underline{\mathbf{I}}}^n\right)^{-1} \left(d^n_{Ap} \times \underline{G}^n_{AB}\right) \tag{2.290}$$

from which it follows that

$$\underline{d}^{n+1}_{Ap} = \underline{\tilde{d}}^{n+1}_{Ap} + \frac{\Delta t^2}{2} g^n_{AB} \underline{U}^n_A \times \underline{d}^n_{Ap} \tag{2.291}$$

where we have defined

$$\underline{U}^n_A = \left(\underline{\underline{\mathbf{I}}}^n_A\right)^{-1} \left(\underline{d}^n_{Ap} \times \underline{d}^n_{ABp}\right) \tag{2.292}$$

and we have used the identity

$$\underline{G}^n_{AB} = g^n_{AB} \underline{d}^n_{ABp}$$

where g^n_{AB} is a scalar quantity. Now, the true position (at timestep t_{n+1}) of the link atom on rigid body A is

$$\underline{r}^{n+1}_{Ap} = \underline{R}^{n+1}_A + \underline{d}^{n+1}_{Ap} \tag{2.293}$$

and inserting (2.288) and (2.291) leads to

$$\underline{r}^{n+1}_{Ap} = \underline{\tilde{R}}^{n+1}_A + \underline{\tilde{d}}^{n+1}_{Ap} + g^n_{AB} \frac{\Delta t^2}{2} \underline{\Theta}_A \tag{2.294}$$

where

$$\underline{\Theta}_A = \left(\frac{\underline{d}^n_{ABp}}{M_A} + \underline{U}^n_A \times \underline{d}^n_{Ap} \right). \tag{2.295}$$

Since $\underline{d}^{n+1}_{ABp} = \underline{r}^{n+1}_{Ap} - \underline{r}^{n+1}_{Bp}$ we can easily obtain

$$\underline{d}^{n+1}_{ABp} = \underline{\tilde{d}}^{n+1}_{ABp} + g^n_{AB} \frac{\Delta t^2}{2} (\underline{\Theta}_A - \underline{\Theta}_B) \tag{2.296}$$

Squaring both sides and neglecting terms of order higher than $O(\Delta t^2)$ gives after rearrangement:

$$g^n_{AB} \approx \frac{(d^{n+1}_{ABp})^2 - (\tilde{d}^{n+1}_{ABp})^2}{\Delta t^2 \underline{\tilde{d}}^{n+1}_{ABp} \cdot (\underline{\Theta}_A - \underline{\Theta}_B)} \tag{2.297}$$

From which the constraint force may be calculated. Iteration is necessary as in SHAKE.

In the second stage of QSHAKE we need to calculate another constaint force $\underline{H}^{n+1}_{AB}$ to preserve the orthogonality of the constraint bond vector and the relative velocity of the two atoms in the bond. Once again the contraint force implies corrections to the translational

and rotational equations of motion, which, following the methods used above, we write directly as:

$$\begin{aligned} \underline{V}_A^{n+1} &= \underline{\tilde{V}}_A^{n+1} + \frac{\Delta t}{2M_A}\underline{H}_{AB}^{n+1} \\ \underline{\omega}_A^{n+1} &= \underline{\tilde{\omega}}_A^{n} + \frac{\Delta t}{2}h_{AB}^{n+1}\underline{U}_A^{n+1} \end{aligned} \tag{2.298}$$

where h_{AB}^{n+1} is a scalar related to the constraint force via

$$\underline{H}_{AB}^{n+1} = h_{AB}^{n+1}\underline{d}_{ABp}^{n+1}$$

Now, the velocity of the linked atom on molecule A is:

$$\underline{v}_{Ap}^{n+1} = \underline{V}_A^{n+1} + \underline{\omega}_A^{n+1} \times \underline{d}_{Ap}^{n+1} \tag{2.299}$$

which on substitution of the above equations gives:

$$\underline{v}_{Ap}^{n+1} = \underline{\tilde{v}}_{Ap}^{n+1} + \frac{\Delta t}{2}h_{AB}^{n+1}\underline{\Omega}_A^{n+1} \tag{2.300}$$

where

$$\underline{\Omega}_A^{n+1} = \left(\frac{\underline{d}_{ABp}^{n+1}}{M_A} + \underline{U}_A^{n+1}\right). \tag{2.301}$$

The constraint condition requires that

$$\underline{d}_{ABp}^{n+1} \cdot (\underline{v}_{Ap}^{n+1} - \underline{v}_{Bp}^{n+1}) = 0 \tag{2.302}$$

and substitution of the equation for $\underline{v}_{Ap}^{n+1}$ (and the equivalent for $\underline{v}_{Bp}^{n+1}$) leads directly to

$$h_{AB}^{n+1} = -\frac{2d_{ABp}^{n+1} \cdot (\underline{\tilde{v}}_{Ap}^{n+1} - \underline{\tilde{v}}_{Bp}^{n+1})}{\Delta t \underline{d}_{ABp}^{n+1} \cdot (\underline{\Omega}_A - \underline{\Omega}_B)} \tag{2.303}$$

which provides the correction for second constraint. This again requires iteration.

The VV QSHAKE algorithm is implemented in DL_POLY_2 in subroutine NVEQVV_2 with the QSHAKE constraint forces calculated in QRATTLE_R and QRATTLE_V. Again it is straightforward to couple these systems to a Hoover or Berendsen thermostat and/or barostat. The Hoover and Berendsen thermostated versions are found in NVTQVV_H2 and NVTQVV_B2 respectively. The isotropic constant pressure implementations are found in NPTQVV_H2 and NPTQVV_B2, while the anisotropic constant pressure routines are found in NSTQVV_H2 and NSTQVV_B2. The Hoover versions make use of the thermostat and barostat routines NVTQSCL, NPTQSCL_T, NPTQSCL_P, NSTQSCL_T and NSTQSCL_P according to the ensemble.

The LF QSHAKE algorithm is implemented in NVEQ_2 with the QSHAKE constraint forces applied in QSHAKE. This also has different ensemble versions: Hoover or Berendsen thermostat and/or barostat. The Hoover and Berendsen thermostated versions are found in NVTQ_H2 and NVTQ_B2 respectively. The isotropic constant pressure implementations are found in NPTQ_H2 and NPTQ_B2, while the anisotropic constant pressure routines are found in NSTQ_H2 and NSTQ_B2.

An outline of the parallel version of QSHAKE is given in section 2.6.9.

2.5.8 The DL_POLY_2 Multiple Timestep Algorithm

For simulations employing a large spherical cutoff r_{cut} in the calculation of the interactions DL_POLY_2 offers the possibility of using a multiple timestep algorithm to improve the efficiency. The method is based on that described by Streett *et al* [46, 47] with extension to Coulombic systems by Forester *et al* [48].

In the multiple timestep algorithm there are two cutoffs for the pair interactions: a relatively large cutoff (r_{cut}) which is used to define the standard Verlet neighbour list; and a smaller cutoff r_{prim} which is used to define a *primary list* within the larger cutoff sphere (see figure). Forces derived from atoms in the primary list are generally much larger than those derived from remaining (so-called *secondary*) atoms in the neighbour list. Good energy conservation is therefore possible if the forces derived from the primary atoms are calculated *every* timstep, while those from the secondary atoms are calculated much less frequently, and are merely extrapolated over the interval. DL_POLY_2 handles this procedure as follows.

DL_POLY_2 updates the Verlet neighbour list at irregular intervals, determined by the movement of atoms in the neighbour list (see section 2.1). The interval between updates is usually of the order of ∼20 timesteps. Partitioning the Verlet list into primary and secondary atoms always occurs when the Verlet list is updated, and thereafter at intervals of `multt` timesteps (i.e. the multi-step interval specified by the user - see section 4.1.1). Immediately after the partitioning, the force contributions from both the primary and secondary atoms are calculated. The forces are again calculated *in total* in the subsequent timestep. Thereafter, for `multt`-2 timesteps, the forces derived from the primary atoms are calculated explicitly, while those derived from the secondary atoms are calculated by linear extrapolation of the exact forces obtained in the first two timesteps of the multi-step interval. It is readily apparent how this scheme can lead to a significant saving in execution time.

Extension of this basic idea to simulations using the Ewald sum requires the following:

1. the reciprocal space terms are calculated only for the first two timesteps of the multi-step,

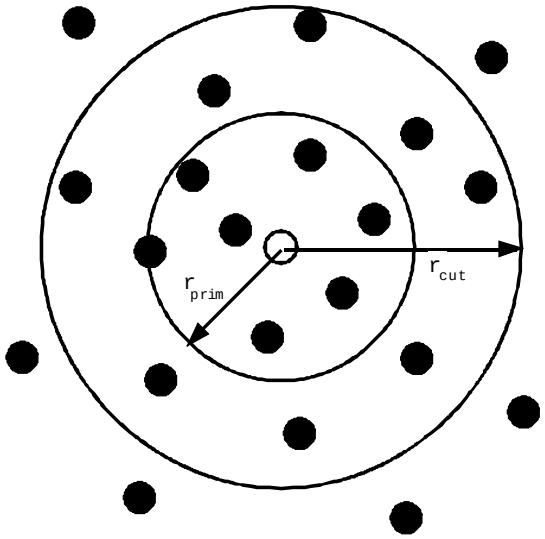
2. the contribution to the reciprocal space terms arising from primary interactions are immediately subtracted, leaving only the long-range components. (This is done in real space, by subtracting *erf* terms.);

3. the real space Coulombic forces arising from the secondary atoms are calculated in the first two timesteps of the multi-step using the normal Ewald expressions (i.e. the *erfc* terms).

4. the Coulombic forces arising from primary atoms are calculated at every timestep in real space assuming the *full Coulombic force*;

In this way the Coulombic forces can be handled by the same multiple timestep scheme as the van der Waals forces. The algorithm is described in detail in [48].

Note that the accuracy of the algorithm is a function of the multi-step interval `multt`, and decreases as `multt` increases. Also, the algorithm is *not time reversible* and is therefore susceptible to energy drift. Its use with a thermostat is therefore advised.



The multiple timestep algorithm

The atoms surrounding the central atom (open circle) are classified as primary if they occur within a radius r_{prim} and secondary if outside this radius but within r_{cut}. Interactions arising from primary atoms are evaluated every timestep. Interactions from secondary atoms are calculated exactly for the first two steps of a multi-step and by extrapolation afterwards.

2.6 DL_POLY Parallelisation

DL_POLY_2 is a distributed parallel molecular dynamics package based on the Replicated Data parallelisation strategy [49, 50]. In this section we briefly outline the basic methodology. Users wishing to add new features DL_POLY_2 will need to be familiar with the underlying techniques as they are described (in greater detail) in references [37, 50]).

2.6.1 The Replicated Data Strategy

The Replicated Data (RD) strategy [49] is one of several ways to achieve parallelisation in MD. Its name derives from the replication of the configuration data on each node of a parallel computer (i.e. the arrays defining the atomic coordinates $\underline{r}_i$, velocities $\underline{v}_i$ and forces $\underline{f}_i$, for all N atoms $\{i : i = 1, \ldots, N\}$ in the simulated system, are reproduced on every processing node). In this strategy most of the forces computation and integration of the equations of motion can be shared easily and equally between nodes and to a large extent be processed independently on each node. The method is relatively simple to program and is reasonably efficient. Moreover, it can be "collapsed" to run on a single processor very

easily. However the strategy can be expensive in memory and have high communication overheads, but overall it has proven to be successful over a wide range of applications. These issues are explored in more detail in [49, 50].

Systems containing complex molecules present several difficulties. They often contain ionic species, which usually require Ewald summation methods [11, 51], and *intra*-molecular interactions in addition to *inter*-molecular forces. These are handled easily in the RD strategy, though the SHAKE algorithm [12] requires significant modification [37].

The RD strategy is applied to complex molecular systems as follows:

1. Using the known atomic coordinates $\underline{r}_i$, each node calculates a subset of the forces acting between the atoms. These are usually comprised of:
   (a) atom-atom pair forces (e.g. Lennard Jones, Coulombic etc.);
   (b) non-rigid atom-atom bonds;
   (c) valence angle forces;
   (d) dihedral angle forces;
   (e) improper dihedral angle forces.
2. The computed forces are accumulated in (incomplete) atomic force arrays $\underline{f}_i$ independently on each node;
3. The atomic force arrays are summed globally over all nodes;
4. The complete force arrays are used to update the atomic velocities and positions.

It is important to note that load balancing (i.e. equal and concurrent use of all processors) is an essential requirement of the overall algorithm. In DL_POLY_2 this is accomplished for the pair forces with an adaptation of the Brode-Ahlrichs scheme [22].

2.6.2 Distributing the Intramolecular Bonded Terms

DL_POLY_2 handles the intramolecular in which the atoms involved in any given bond term are explicitly listed. Distribution of the forces calculations is accomplished by the following scheme:

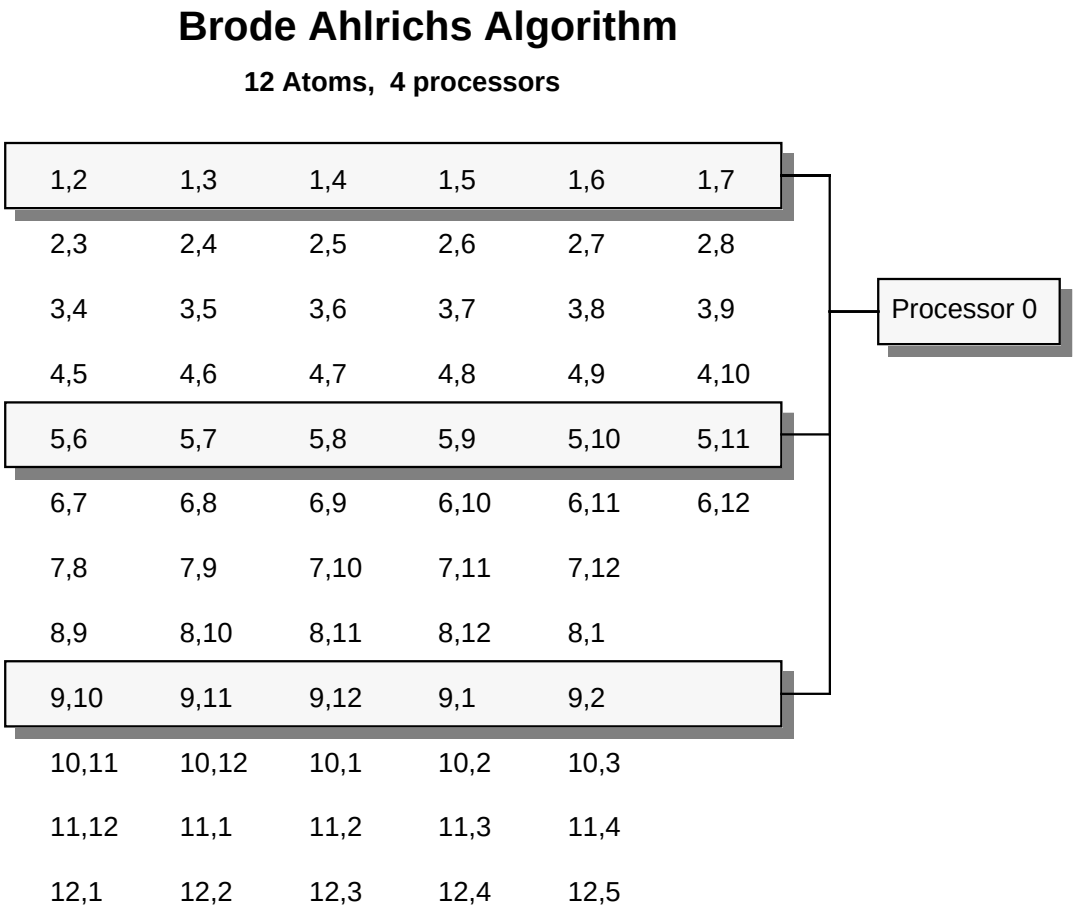
1. Every atom in the simulated system is assigned a unique index number from 1 to N;
2. Every intramolecular bonded term U_{type} in the system has a unique index number i_{type}: from 1 to N_{type} where *type* represents a bond, angle or dihedral.
3. A pointer array $key_{type}(n_{type}, i_{type})$ carries the indices of the specific atoms involved in the potential term labelled i_{type}. The dimension n_{type} will be 2, 3 or 4, if the term represents a bond, angle or dihedral.
4. The array $key_{type}(n_{type}, i_{type})$ is used to identify the atoms in a bonded term and the appropriate form of interaction and thus to calculate the energy and forces. Each processor is assigned the independent task of evaluating a block of $(Int(N_{total}/N_{nodes}))$ interactions.

The same scheme works for all types of bonded interactions. The global summation of the force arrays does not occur until all the force contributions, including nonbonded forces has been completed.

2.6.3 Distributing the Nonbonded Terms

In DL_POLY_2 the nonbonded interactions are handled with a Verlet neighbour list [11] which is reconstructed at intervals during the simulation. This list records the indices of all 'secondary' atoms within a certain radius of each 'primary' atom; the radius being the cut-off radius (r_{cut}) normally applied to the nonbonded potential function, plus an additional increment (Δr_{cut}). The larger radius ($r_{cut} + \Delta r_{cut}$) permits the same list to be used for several timesteps without requiring an update. The frequency at which the list must be updated clearly depends on the thickness of the region Δr_{cut}. In RD, the neighbour list is constructed *simultaneously* on each node and in such a way as to share the total burden of the work equally between nodes. Each node is responsible for a unique set of nonbonded interactions and the neighbour list is therefore different on each node. DL_POLY_2 uses a method based on the Brode-Ahlrichs scheme [22] (see figure below) to construct the neighbour list.

Additional modifications are necessary to handle the excluded atoms [50]. A distributed *excluded atoms list* is constructed by DL_POLY_2 at the start of the simulation. The list is constructed so that the excluded atoms are referenced in the same order as they would appear in the Verlet neighbour list if the bonded interactions were ignored, allowing for the distributed structure of the neighbour list.



Schematic diagram of the parallel implementation of the Brode-Ahlrichs algorithm.

The diagram illustrates the reordering of the upper triangular matrix of n(n-1)/2 pair interactions so that the rows of the matrix are of approximately equally length. Each entry in the table consists of a primary atom index (constant within a row) and a "neighbouring" atom index. Rows are assigned sequentially to nodes. In the diagram node 0 deals with rows 1, 5 and 9, node 1 to rows 2, 6, and 10 etc.

When a charge group scheme (as opposed to an atomistic scheme) is used for the non-bonded terms, the group-group interactions are distributed using the Brode-Ahlrichs approach. This makes the Verlet list considerably smaller, thus saving memory, but also results in a more "coarse grain" parallelism. The consequence of which is that performance with a large number of processors will degrade more quickly than with the atomistic scheme.

Once the neighbour list has been constructed, each node of the parallel computer may proceed independently to calculate the pair force contributions to the atomic forces.

2.6.4 Modifications for the Ewald Sum

For systems with periodic boundary conditions DL_POLY_2 employs the Ewald Sum to calculate the Coulombic interactions (see section 2.4.5).

Calculation of the real space component in DL_POLY_2 employs the algorithm for the calculation of the nonbonded interactions outlined above. The reciprocal space component is calculated using the schemes described in [51], in which the calculation can be parallelised by distribution of either $\underline{k}$ vectors or atomic sites. Distribution over atomic sites requires the use of a global summation of the $q_i \exp(-i\underline{k} \cdot \underline{r}_j)$ terms, but is more efficient in memory usage. Both strategies are computationally straightforward. Subroutine EWALD1 distributes over atomic sites and is often the more efficient of the two approaches. Subroutine EWALD1A distributes over the $\underline{k}$ vectors and may be more efficient on machines with large communication latencies.

Other routines required to calculate the ewald sum include EWALD2, EWALD3 and EWALD4. The first of these calculates the real space contribution, the second the self interaction corrections, and the third is required for the multiple timestep option.

2.6.5 Modifications for SPME

The SPME method requires relatively little modification for parallel computing. The real space terms are calculated exactly as they are for the normal Ewald sum, as described above. The reciprocal space sum requires a 3D Fast Fourier Transform (FFT), which in principle should be distributed over the processors, but in DL_POLY_2 the decision was made to implement a complete 3D FFT on every processor. This is expensive in memory, and potentially expensive in computer time. However a multi-processor FFT requires communication between processors and this has significant impact on the famed efficiency of the FFT. It transpires that a single processor FFT is so efficient that the adopted strategy is still effective. The charge array that is central to the SPME method (see section 2.4.6) is however built in a distributed manner and then globally summed prior to the FFT operation.

2.6.6 Three and Four Body Forces

DL_POLY_2 can calculate three/four body interactions of the valence angle type [52]. These are not dealt with in the same way as the normal nonbonded interactions. They are generally very short ranged and are most effectively calculated using a link-cell scheme [23]. No reference is made to the Verlet neighbour list nor the excluded atoms list. It follows that atoms involved in the same three/four-body term can interact via nonbonded (pair) forces and ionic forces also. The calculation of the three/four-body terms is distributed over processors on the basis of the identity of the central atom in the bond. A global summation is required to specify the atomic forces fully.

2.6.7 Metal Potentials

The simulation of metals by DL_POLY_2 makes use of density dependent potentials of the Sutton-Chen type [3]. The dependence on the atomic density presents no difficulty however, as this class of potentials can be resolved into pair contributions. This permits the use of the distributed Verlet neighbour list outlined above.

2.6.8 Summing the Atomic Forces

The final stage in the RD strategy, is the global summation of the atomic force arrays. This must be done After all the contributions to the atomic forces have been calculated. To do this DL_POLY_2 employs a global summation algorithm [49], which is generally a system specific utility.

Similarly, the total configuration energy and virial must be obtained as a global sum of the contributing terms calculated on all nodes.

2.6.9 The SHAKE, RATTLE and Parallel QSHAKE Algorithms

The SHAKE and RATTLE algorithms are methods for constraining rigid bonds. Parallel adaptations of both are couched in the Replicated Data strategy. The essentials of the methods are as follows.

1. The bond constraints acting in the simulated system are shared equally between the processing nodes.
2. Each node makes a list recording which atoms are bonded by constraints it is to process. Entries are zero if the atom is not bonded.
3. A copy of the array is passed to each other node in turn. The receiving node compares the incoming list with its own and keeps a record of the shared atoms and the nodes which share them.
4. In the first stage of the SHAKE algorithm, the atoms are updated through the usual Verlet algorithm, without regard to the bond constraints.
5. In the second (iterative) stage of SHAKE, each node calculates the incremental correction vectors for the bonded atoms in its own list of bond constraints. It then sends specific correction vectors to all neighbours that share the same atoms, using the information compiled in step 3.
6. When all necessary correction vectors have been received and added the positions of the constrained atoms are corrected.
7. Steps 5 and 6 are repeated until the bond constraints are converged.
8. After convergence the coordinate arrays on each node are passed to all the other nodes. The coordinates of atoms that are *not* in the constraint list of a given node are taken from the incoming arrays (an operation we term *splicing*).
9. Finally, the change in the atom positions is used to calculate the atomic velocities.

The above scheme is complete for a implementation based on the leapfrog integration algorithm. However a velocity Verlet (VV) scheme requires additional steps.

1. Step 9 above does not apply for VV. The velocity is integrated under the normal VV scheme.

2. When the velocity is updated, iteration of the constraint force takes place. The incremental changes to the velocity are communicated between nodes sharing constrained atoms as for the bondlength constraints.

3. Iteration is repeated until the bond constraints are converged.

4. After convergence the velocity arrays on each node are passed to all the other nodes by *splicing*.

This scheme contains a number of non-trivial operations, which are described in detail in [37]. However some general comments are worth making.

The compilation of the list of constrained atoms on each node, and the circulation of the list (items 1 - 3 above) need only be done once in any given simulation. It also transpires that in sharing bond contraints between nodes, there is an advantage to keeping as many of the constraints pertaining to a particular molecule together on one node as is possible within the requirement for load balancing. This reduces the data that need to be transferred between nodes during the iteration cycle. It is also advantageous, if the molecules are small, to adjust the load balancing between processors to prevent shared atoms. The loss of balance is compensated by the elimination of communications during the SHAKE cycle. These techniques are exploited by DL_POLY_2 .

The QSHAKE algorithm is an extension of the SHAKE algorithm for constraint bonds between rigid bodies. The parallel strategy is very similar to that of SHAKE. The only significant difference is that increments to the atomic forces, not the atomic positions, are passed between processors at the end of each iteration.

Chapter 3

DL_POLY_2 Construction and Execution

Scope of Chapter

This chapter describes how to compile a working version of DL_POLY_2 and how to run it.

3.1 Constructing DL_POLY_2 : an Overview

3.1.1 Constructing the Standard Version

DL_POLY_2 was designed as a package of useful subroutines rather than a single program, which means that users are to be able to construct a working simulation program of their own design from the subroutines available, which is capable of performing a specific simulation. However we recognise that many, perhaps most, users will be content with creating a standard version that covers all of the possible applications and for this reason we have provided the necessary tools to assemble such a version. The method of creating the standard version is described in detail in this chapter, however a brief step-by-step description follows.

1. DL_POLY_2 is supplied as a UNIX compressed tar file. This must uncompressed and un-tared to create the DL_POLY_2 directory (section 1.4).

2. In the *build* subdirectory you will find the required DL_POLY_2 makefile (see section 3.2.1 and Appendix A, where a sample Makefile is listed). This must be copied into the subdirectory containing the relevant source code. In most cases this will be the *srcf90* subdirectory.

3. The makefile is executed with the appropriate keywords (section 3.2.1) which select for specific computers and if a parallel machine is used, the appropriate communication software.

4. The makefile produces the executable version of the code, which as a default will be named DLPOLY.X and located in the *execute* subdirectory.

5. DL_POLY also has a Java GUI. The files for this are stored in the subdirectory *java*. Compilation of this is simple and requires running the javac compiler and the jar utility. Details for these procedures are provided in the GUI manual [8].

6. To run the executable for the first time you require the files CONTROL, FIELD and CONFIG (and possibly TABLE if you have tabulated potentials). These must be present in the directory from which the program is executed. (See section 4.1 for the description of the input files.)

7. Executing the program will produce the files OUTPUT, REVCON and REVIVE (and optionally STATIS, HISTORY, RDFDAT and ZDNDAT) in the executing directory. (See section 4.2 for the description of the output files.)

This simple procedure is enough to create a standard version to run most DL_POLY_2 applications. However it sometimes happens that additional modifications may be necessary.

On starting, DL_POLY_2 scans the input data and makes an estimate of the sizes of the arrays it requires to do the simulation. Sometimes the estimates are not good enough. The most common occurrences of this are NPT and NST simulations, or simulations where the

local density on the MD cell may significantly exceed the mean density of the cell (systems with a vaccum gap for example). Under these circmstances arrays initally allocated may be insufficent. In which case DL_POLY_2 may report a memory problem and request that you recompile the code with hand-adjusted array dimensions. This topic is dealt with more fully in Appendix C.

3.1.2 Constructing Nonstandard Versions

In constructing a nonstandard DL_POLY_2 simulation program, the first requirement is for the user to write a program to function as the root segment. The *srcf90* directory contains an example of such a root program: DLPOLY. This root program calls the major routines required to perform the simulation and also controls the normal "molecular dynamics cycle" which consists of forces calculation followed by integration of the equation of motion. DLPOLY also monitors the *cpu* usage and brings about a controlled termination of the program if the usage approaches the allotted job time within a pre-set closure time. Lastly, DLPOLY is the routine that first opens the OUTPUT file (section 4.2.2), which provides the summary of the job. Users are recommended to study the DLPOLY root as a model for other implementations of the package they may wish to construct.

If additional functionality is added to DL_POLY_2 by the user, the PARSET.F subroutine (and its support subroutines) will need modifying to allow specification of the dimensions of any new arrays.

Any molecular dynamics simulation performs five different kinds of operation: initialisation; forces calculation; integration of the equations of motion; calculation of system properties; and job termination. It is worth considering these operations in turn and to indicate which DL_POLY_2 routines are available to perform them. We do not give a detailed description, but provide only a guide. The following outline assumes a system containing flexible molecules held together by rigid bonds, but without rigid bodies.

Initialisation requires firstly that the program determine what kind of parallel machine it is running on. The routine MACHINE determines how many processing nodes are being used and also returns the node identity to each process. Next the job control information is required; this is obtained by the routine SIMDEF, which reads the CONTROL file (section 4.1.1). The description of the system to be simulated: the types of atoms and molecules present and the intermolecular forces; are obtained by the SYSDEF routine, which reads the FIELD file (section 4.1.3). Lastly, the atomic positions and velocities must be provided. These are obtained by the SYSGEN routine, which reads the CONFIG file (section 4.1.2) and also generates the initial velocities if required to do so. If the system contains constraint bonds, the routine PASSCON is required to process molecular connectivity data and establish the communication procedure between nodes, and the QUENCH routine is required to set the starting velocities correctly. Also needed in the initialisation, is the routine FORGEN, which constructs the interpolation arrays for the short-range forces calculations, and the routine EXCLUDE which identifies atoms that are explictly chemically bonded through bonds, constraints or valence angles. The resulting list is known as the *excluded atoms* list.

The calculation of the pair forces represents the bulk of any simulation. A Verlet neighbour list is used by DL_POLY_2 in calculating the atomic forces. The routine that

constructs this this is called PARLST. This routine builds the neighbour list taking into account the occurrence of atoms in the *excluded atoms* list. The routine SRFRCE calculates the short-range (van der Waals) forces, making use of the IMAGES routine to handle any periodic boundary conditions. Coulombic forces are handled by a varity of routines: COUL0, COUL1 and COUL2 handle Coulombic forces *without* periodic boundaries; EWALD1, EWALD2 and EWALD3 are used for systems *with* periodic boundaries (an additional routine: EWALD4 is necessary for the multiple timestep algorithm). Intramolecular forces require the routines ANGFRC, BNDFRC and DIHFRC. If the multiple timestep algorithm is required, the routine MULTIPLE must be used to call the various forces routines. It also calls the PRIMLST routine to split the interaction list into primary and secondary neighbours. The decision to update the neighbour list is handled by the routine VERTEST. The routine EXTNFLD is required if the simulated system has an external force field (e.g. electrostatic field) operating. To help with equilibration simulations, the routine FCAP is sometimes required to reduce the magnitude of badly equilibrated forces. Since DL_POLY_2 is based on the replicated data strategy, a global sum routine (GDSUM) is required to sum the atomic forces on all nodes.

Integration of the equations of motion is handled by one of the routines listed and described in section 2.5. For example routines NVE_0, NVT_E0, NVT_H0, NVT_B0 etc. are used if no constraint forces are present. These routines treat the NVE, Evans-NVT, Hoover-Nosé-NVT and NVT-Berendsen ensembles respectively. The corresponding versions of these routines which handle constraint forces are NVE_1, NVT_E1, NVT_H1 or NVT_B1. These versions call the routine RDSHAKE_1 to handle the constraints. RDSHAKE_1 itself calls a number of additional routines: MERGE, SHMOVE and SPLICE. For *ad hoc* temperature scaling, the routine VSCALEG is required.

As mentioned elsewhere, DL_POLY_2 does not contain many routines for computing system properties during a simulation. Radial distributions may be calculated however, using the routines RDF0 and RDF1. Similarly DIFFSN0 and DIFFSN1 calculate approximate mean square displacements. Ordinary thermodynamic quantities are calculated by the routine STATIC, which also writes the STATIS file (section 4.2.7). Routine TRAJECT writes the HISTORY (section 4.2.1) file for later analysis.

Job termination is handled by the routine RESULT which writes the final summaries in the OUTPUT file and dumps the restart files REVIVE and REVCON (sections 4.2.4 and 4.2.3 respectively).

An idea of the construction of a DL_POLY_2 program can be obtained from the following flowchart. The example represents a DL_POLY_2 program which uses the multiple timestep algorithm, with bond constraints and the Nosé-Hoover thermostat.

3.2 Compiling and Running DL_POLY_2

3.2.1 Compiling the Source Code

When you have obtained DL_POLY_2 from Daresbury Laboratory and unpacked it, your next task will be to compile it. To aid compilation a set of makefiles has been provided in the sub-directory *build* (see example in Appendix A of this document). The versions go by the names of:

- MakePAR - to build a parallel MPI version on a unix platform;
- MakeSEQ - to build a sequential (one processor) unix version;
- MakeWIN - to build a Windows (one processor, XP) version.

Select the one you need and copy it into the *srcf90* directory. (In what follows we assume the makefile in the *srcf90* directory is called 'Makefile'.) The Makefile will build an executable with a wide range of functionality - sufficient for the test cases and for most users' requirements. Other makefiles may be found in the *build* sub-directory for variants of DL_POLY_2 .

Users will need to modify the Makefile if they are to add additional functionality to the code, or if it requires adaptation for a non specified computer. Modifications may also be needed for the Smoothed Particle Mesh Ewald method if a system specific 3D FFT routine is desired (see below: "Modifying the makefile").

Note the following system requirements for a successful build of DL_POLY_2 .

1. a FORTRAN 90 compiler;
2. the Java SDK from Sun Microsystems (if the GUI is required).
3. a UNIX operating system (or Windows XP with CygWin, if a PC version is required).

Run the Makefile you copied from the *build* sub-directory in the *srcf90* sub-directory. It will create the executable in the *execute* sub-directory. The compilation of the program is initiated by typing the command:

make target

where *target* is the specification of the required machine (e.g. hpcx). For many computer systems this is all that is required to compile a working version of DL_POLY_2 . (To determine which targets are already defined in the makefile, typing the command *make* without a nominated target will produce a list of known targets.)

The full specification of the *make* command is as follows

make <TARGET= . . . > < EX=. . . > < BINROOT=. . . >

where some (or all) of the keywords may be omitted. The keywords and their uses are described below. Note that keywords may also be set in the unix environment (e.g. with the "setenv" command in a C-shell).

For PCs running Windows, the makefile assumes the user has installed the Cygwin Unix API available from http://sources.redhat.com/cygwin. The recommended FORTRAN 90 compiler is G95 (see: http://ftp.g95.org/). Both of these are copyrighted products.

3.2.1.1 Keywords for the Makefile

1. **TARGET**

   The TARGET keyword indicates which kind of computer the code is to be compiled for. This **must** be specifed - there is no default value. Valid targets can be listed by the makefile if the command *make* is typed, without arguments. The list frequently changes as more targets are added and redundant ones removed. Users are encouraged to extend the Makefile for themselves, using existing targets as examples.

2. **EX**

   The EX keyword specifies the executable name. The default name for the executable is "DLPOLY.X".

3. **BINROOT**

   The BINROOT keyword specifies the directory in which the executable is to be stored. The default setting is "../execute".

3.2.1.2 Modifying the Makefile

1. **Changing the TARGET**

   If you do not intend to run DL_POLY_2 on one of the specified machines, you must add appropriate lines to the makefile to suit your circumstances. The safest way to do this is to modify an existing TARGET option for your purposes. The makefile supplied with DL_POLY_2 contains examples for serial and MPI environments as well as for different parallel machines, so you should find one close to your requirements. You must of course be familiar with the appropriate invocation of the FORTRAN compiler for your local machine and also any alternatives to MPI your local machine may be running. If you wish to compile for MPI systems remember to ensure the appropriate library directories are accessible to you. If you require a serial version of the code, you must remove references to the MPI libraries from the Makefile and add the file serial.f to your compilation - this will insert replacement (dummy) routiens for the MPI calls.

2. **Enabling the Smoothed Particle Mesh Ewald**

The standard compilation of DL_POLY_2 will incorporate a basic 3D Fast Fourier Transform (FFT) routine to enable the SPME functionality. Users may wish to try alternative FFT routines, which may offer faster performance. Some 'hooks' for these appear in the code as comment lines in the FORTRAN source. The user should search for the following keys in the code:

- CCRAY - for the Cray FFT routines;
- CFFTW - for the FFTW public domain FFT routines;
- CESSL - for the IBM scientific library FFT routines;
- CSGIC - for the Silicon Graphics FFT routines.

The appropriate lines should be uncommented and the references to the DLPFFT3 subroutine should be commented out before compiling.

3. **Problems with optimization ?**

   Some subroutines may not compile correctly when using optimization on some compilers. This is not the fault of the DL_POLY_2 code, but of the compiler concerned. This is circumvented by compiling the offending subroutines unoptimised. See the entries for various machines in the makefile to see how this is done if you experience problems with other subroutines.

4. **Adding new functionality**

   To include a new subroutine in the code simply add *subroutine*.o to the list of object names in the makefile. The simplest way is to add names to the "OBJ_ALL" list.

3.2.1.3 Note on Interpolation

In DL_POLY_2 the short-range (Van der Waals) contributions to energy and force are evaluated by interpolation of tables constructed at the beginning of execution. DL_POLY_2 employs a 3-point interpolation scheme.

A guide to the *minimum* number of grid points (`mxgrid`) required for interpolation in r to give good energy conservation in a simulation is:

$$\texttt{mxgrid} \geq 100(\texttt{rcut}/\texttt{rmin})$$

where `rmin` is the *smallest* position minimum of the non-bonded potentials in the system. The parameter `mxgrid` is defined in the DL_PARAMS.INC file, and must be set before compilation.

A utility program TABCHK is provided in the DL_POLY *utility* sub-directory to help users choose a sufficiently accurate interpolation scheme (including array sizes) for their needs.

3.2.2 Running DL_POLY_2

To run the DL_POLY_2 executable (DLPOLY.X) you will initially require three, possibly four, input data files, which you must create in the *execute* sub-directory, (or whichever sub-directory you keep the executable program.) The first of these is the CONTROL file (section 4.1.1), which indicates to DL_POLY_2 what kind of simulation you want to run, how much data you want to gather and for how long you want the job to run. The second file you need is the CONFIG file (section 4.1.2). This contains the atom positions and, depending on how the file was created (e.g. whether this is a configuration created from 'scratch' or the end point of another run), the velocities also. The third file required is the FIELD file (section 4.1.3), which specifies the nature of the intermolecular interactions, the molecular topology and the atomic properties, such as charge and mass. Sometimes you will require a fourth file: TABLE (section 4.1.5), which contains the potential and force arrays for functional forms not available within DL_POLY_2 (usually because they are too complex e.g. spline potentials).

Examples of input files are found in the *data* sub-directory, which can be copied into the *execute* subdirectory using the *select* macro found in the *execute* sub-directory.

A successful run of DL_POLY_2 will generate several data files, which appear in the *execute* sub-directory. The most obvious one is the file OUTPUT (section 4.2.2), which provides an effective summary of the job run: the input information; starting configuration; instantaneous and rolling-averaged thermodynamic data; final configurations; radial distribution functions (RDFs); and job timing data. The OUTPUT file is human readable. Also present will be the restart files REVIVE (section 4.2.4) and REVCON (section 4.2.3). REVIVE contains the accumulated data for a number of thermodynamic quantities and RDFs, and is intended to be used as the input file for a following run. It is *not* human readable. The REVCON file contains the *restart configuration* i.e. the final positions, velocities and forces of the atoms when the run ended and is human readable. The STATIS file (section 4.2.7) contains a catalogue of instantaneous values of thermodynamic and other variables, in a form suitable for temporal or statistical analysis. Finally, the HISTORY file (section 4.2.1) provides a time ordered sequence of configurations to facilitate further analysis of the atomic motions. Depending on which version of the TRAJECT subroutine you compiled in the code, this file may be either formatted (human readable) or unformatted. You may move these output files back into the *data* sub-directory using the *store* macro found in the *execute* sub-directory.

Note that versions of DL_POLY_2 after 2.10 may also create the files RDFDAT and ZDNDAT, containing the RDF and Z-density data respectively. They are both human readable files.

3.2.3 Restarting DL_POLY_2

The best approach to running DL_POLY_2 is to define from the outset precisely the simulation you wish to perform and create the input files specific to this requirement. The program will then perform the requested simulation, but may terminate prematurely through error, inadequate time allocation or computer failure. Errors in input data are your responsibil-

ity, but DL_POLY_2 will usually give diagnostic messages to help you sort out the trouble. Running out of job time is common and provided you have correctly specified the job time variables (using the **close time** and **job time** directives - see section 4.1.1) in the CONTROL file, DL_POLY_2 will stop in a controlled manner, allowing you to restart the job as if it had not been interrupted.

To restart a simulation after normal termination you will again require the CONTROL file, the FIELD (and TABLE) file, and a CONFIG file, which is the exact copy of the REVCON file created by the previous job. You will also require a new file: REVOLD (section 4.1.4), which is an exact copy of the previous REVIVE file. If you attempt to restart DL_POLY_2 without this additional file available, the job will fail. Note that DL_POLY_2 will append new data to the existing STATIS and HISTORY files if the run is restarted, other output files will be **overwritten**.

In the event of machine failure, you should be able to restart the job in the same way from the surviving REVCON and REVIVE files, which are dumped at intervals to meet just such an emergency. In this case check carefully that the input files are intact and use the HISTORY and STATIS files with caution - there may be duplicated or missing records. The reprieve processing capabilities of DL_POLY_2 are not foolproof - the job may crash while these files are being written for example, but they can help a great deal. You are advised to keep backup copies of these files, noting the times they were written, to help you avoid going right back to the start of a simulation.

You can also extend a simulation beyond its initial allocation of timesteps, provided you still have the REVCON and REVIVE files. These should be copied to the CONFIG and REVOLD files respectively and the directive **timesteps** adjusted in the CONTROL file to the new total number of steps required for the simulation. For example if you wish to extend a 10000 step simulation by a further 5000 steps use the directive **timesteps 15000** in the CONTROL file and include the **restart** directive. You can use the **restart scale** directive if you want to reset the temperature at the restart, but note that this also resets all internal accumulators (timestep included) to zero.

3.3 A Guide to Preparing Input Files

The CONFIG file and the FIELD file can be quite large and unwieldy particularly if a polymer or biological molecule is involved in the simulation. This section outlines the paths to follow when trying to construct files for such systems. The description of the DL_POLY_2 force field in chapter 2 is essential reading. The various utility routines mentioned in this section are described in greater detail in chapter 6. Many of these have been incorporated into the DL_POLY_2 Graphical User Interface [8] and may be convienently used from there.

3.3.1 Inorganic Materials

The utility GENLAT can be used to construct the CONFIG file for relatively simple lattice structures. Input is interactive. The FIELD file for such systems are normally small and can be constructed by hand. The utility GENLAT.TO constructs the CONFIG file for truncated-octahedral boundary conditions. Otherwise the input of force field data for crystalline

systems is particularly simple, if no angular forces are required (notable exceptions to this are zeolites and silicate glasses - see below). Such systems require only the specification of the atomic types and the necessary pair forces. The reader is referred to the description of the DL_POLY_2 FIELD file for further details (section 4.1.3).

DL_POLY_2 allows the simulation of zeolites and silicate (or other) glasses. Both these materials require the use of angular forces to describe the local structure correctly. In both cases the angular terms are included as *three body terms*, the forms of which are described in chapter 2. These terms are entered into the FIELD file with the pair potentials. Note that you cannot use truncated octahedral or rhombic dodecahedral boundary conditions in conjunction with three body forces, due to the use of the link-cell algorithm for evaluating the forces.

An alternative way of handling zeolites is to treat the zeolite framework as a kind of macromolecule (see below). Specifying all this is tedious and is best done computationally: what is required is to determine the nearest image neighbours of all atoms and assign appropriate bond and valence angle potentials. (This may require the definition of new bond forces in subroutine BNDFRC, but this is easy.) What must be avoided at all costs is specifying the angle potentials *without* specifying bond potentials. In this case DL_POLY_2 will automatically cancel the non-bonded forces between atoms linked via valence angles and the system will collapse. The advantage of this method is that the calculation is likely to be faster using three-body forces. This method is not recommended for amorphous systems.

3.3.2 Macromolecules

Simulations of proteins are best tackled using the package DLPROTEIN [53] which is an adaptation of DL_POLY specific to protein modelling. However you may simulate proteins and other macromolecules with DL_POLY_2 if you wish. This is described below.

If you select a *protein* structure from a SEQNET file (e.g. from the Brookhaven database), use the utility PROSEQ to generate the file CONFIG. This will then function as input for DL_POLY_2 . Some caution is required here however, as the protein structure may not be fully determined and atoms may be missing from the CONFIG file.

If you have the "edit.out" file produced by AMBER for your molecule use this as the CONNECT_DAT input file for the utility AMBFORCE. AMBFORCE will produce the DL_POLY_2 FIELD and CONFIG files for your molecule.

If you do not have the "edit.out" file things are a little more tricky, particularly in coming up with appropriate partial charges for atomic sites. However there are a series of utilities that will at least produce the CONNECT_DAT file for use with AMBFORCE. We now outline these utilities and the order in which they should be used.

If you have a structure from the Cambridge Structural database (CSDB) then use the utility FRACCON to take fractional coordinate data and produce a CONNECT_DAT and "ambforce.dat" file for use with AMBFORCE. Note that you will need to modify FRACCON to get the AMBER names correct for sites in your molecule. The version of FRACCON supplied with DL_POLY_2 is specific to the valinomycin molecule.

If you require an all atom force field and the database file does not contain hydrogen

positions then use the utility FRACFILL in place of FRACCON. FRACCON produces an output file HFILL which should then be used as input for the utility HFILL. The HFILL utility fills out the structure with the missing hydrogens. (Note that you may need to know what the atomic charges are in some systems, for example the AMBER charges from the literature.)

Note: with minor modifications the utilities FRACFILL and FRACCON can be used on structures from databases other than the Cambridge structural database.

3.3.3 Adding Solvent to a Structure

The utility WATERADD adds water from an equilibrated configuration of 256 SPC water molecules at 300 K to fill out the MD cell. The utility SOLVADD fills out the MD box with single-site solvent molecules from a f.c.c lattice. The FIELD files will then need to be edited to account for the solvent molecules added to the file.

Hint: to save yourself some work in entering the non-bonded interactions variables involving solvent sites to the FIELD file put two bogus atoms of each solvent type at the end of the CONNECT_DAT file (for AMBER force-fields) the utility AMBFORCE will then evaluate all the non-bonded variables required by DL_POLY_2 . Remember to delete the bogus entries from the CONFIG file before running DL_POLY_2 .

3.3.4 Analysing Results

DL_POLY_2 is not designed to calculate every conceivable property you might wish from a simulation. Apart from some obvious thermodynamic quantities and radial distribution functions, it does not calculate anything beyond the atomic trajectories on-line. You must therefore be prepared to post-process the HISTORY file if you want other information. There are some utilities in the DL_POLY_2 package to help with this, but the list is far from exhaustive. In time, we hope to have many more. Our users are invited to submit code to the DL_POLY_2 *public* library to help with this.

The utilities available are described in the DL_POLY_2 Reference Manual, Chapter 6. Users should also be aware that many of these utilities are incorporated into the DL_POLY Graphical User Interface [8].

3.3.5 Choosing Ewald Sum Variables

3.3.5.1 Ewald sum and SPME

This section outlines how to optimise the accuracy of the Ewald sum parameters for a given simulation. In what follows the directive **spme** may be used anywhere in place of the directive **ewald** if the user wishes to use the Smoothed Particle Mesh Ewald method.

As a guide to beginners DL_POLY_2 will calculate reasonable parameters if the **ewald precision** directive is used in the CONTROL file (see section 4.1.1). A relative error (see below) of 10^{-6} is normally sufficient so the directive

ewald precision 1d-6

will cause DL_POLY_2 to evaluate its best guess at the Ewald parameters α, `kmax1`, `kmax2` and `kmax3`. (The user should note that this represents an *estimate*, and there are sometimes circumstances where the estimate can be improved upon. This is especially the case when the system contains a strong directional anisotropy, such as a surface.) These four parameters may also be set explicitly by the **ewald sum** directive in the CONTROL file. For example the directive

ewald sum 0.35 6 6 8

would set $\alpha = 0.35$ Å^{-1}, `kmax1` $= 6$, `kmax2 = 6` and `kmax3` $= 8$. The quickest check on the accuracy of the Ewald sum is to compare the Coulombic energy (U) and the coulombic virial ($\mathcal{W}$) in a short simulation. Adherence to the relationship $U = -\mathcal{W}$ shows the extent to which the Ewald sum is correctly converged. These variables can be found under the columns headed `eng_cou` and `vir_cou` in the OUTPUT file (see section 4.2.2).

The remainder of this section explains the meanings of these parameters and how they can be chosen. The Ewald sum can only be used in a three dimensional periodic system. There are three variables that control the accuracy: α, the Ewald convergence parameter; $r_{\rm cut}$ the real space forces cutoff; and the `kmax1,2,3` integers [1] that effectively define the range of the reciprocal space sum (one integer for each of the three axis directions). These variables are not independent, and it is usual to regard one of them as pre-determined and adjust the other two accordingly. In this treatment we assume that $r_{\rm cut}$ (defined by the **cutoff** directive in the CONTROL file) is fixed for the given system.

The Ewald sum splits the (electrostatic) sum for the infinite, periodic, system into a damped real space sum and a reciprocal space sum. The rate of convergence of both sums is governed by α. Evaluation of the real space sum is truncated at $r = r_{\rm cut}$ so it is important that α be chosen so that contributions to the real space sum are negligible for terms with $r > r_{\rm cut}$. The relative error (ϵ) in the real space sum truncated at $r_{\rm cut}$ is given approximately by

$$\epsilon \approx {\rm erfc}(\alpha r_{\rm cut})/r_{\rm cut} \approx \exp[-(\alpha.r_{\rm cut})^2]/r_{\rm cut} \tag{3.1}$$

The recommended value for α is $3.2/r_{\rm cut}$ or greater (too large a value will make the reciprocal space sum very slowly convergent). This gives a relative error in the energy of no greater than $\epsilon = 4\times10^{-5}$ in the real space sum. When using the directive **ewald precision** DL_POLY_2 makes use of a more sophisticated approximation:

$${\rm erfc}(x) \approx 0.56\exp(-x^2)/x \tag{3.2}$$

to solve recursively for α, using equation 3.1 to give the first guess.

The relative error in the reciprocal space term is approximately

$$\epsilon \approx \exp(-k_{max}^2/4\alpha^2)/k_{max}^2 \tag{3.3}$$

where

$$k_{max} = \frac{2\pi}{L}\,\texttt{kmax} \tag{3.4}$$

[1] **Important note:** For the SPME method the values of `kmax1,2,3` should be double those obtained in this prescription, since they specify the sides of a cube, not a radius of convergence.

is the largest k-vector considered in reciprocal space, L is the width of the cell in the specified direction and `kmax` is an integer.

For a relative error of 4×10^{-5} this means using $k_{max} \approx 6.2\alpha$. `kmax` is then

$$\texttt{kmax} > 3.2\ L/r_{\rm cut} \tag{3.5}$$

In a cubic system, $r_{\rm cut} = L/2$ implies `kmax` $= 7$. In practice the above equation slightly over estimates the value of `kmax` required, so optimal values need to be found experimentally. In the above example `kmax` $= 5$ or 6 would be adequate.

If your simulation cell is a truncated octahedron or a rhombic dodecahedron then the estimates for the `kmax` need to be multiplied by $2^{1/3}$. This arises because twice the normal number of k-vectors are required (half of which are redundant by symmetry) for these boundary contributions [37].

If you wish to set the Ewald parameters manually (via the **ewald sum** or *spme sum* directives) the recommended approach is as follows. Preselect the value of $r_{\rm cut}$, choose a working a value of α of about $3.2/r_{\rm cut}$ and a large value for the `kmax` (say 10 10 10 or more). Then do a series of ten or so *single* step simulations with your initial configuration and with α ranging over the value you have chosen plus and minus 20%. Plot the Coulombic energy (and $-\mathcal{W}$) versus α. If the Ewald sum is correctly converged you will see a plateau in the plot. Divergence from the plateau at small α is due to non-convergence in the real space sum. Divergence from the plateau at large α is due to non-convergence of the reciprocal space sum. Redo the series of calculations using smaller `kmax` values. The optimum values for `kmax` are the smallest values that reproduce the correct Coulombic energy (the plateau value) and virial at the value of α to be used in the simulation.

Note that one needs to specify the three integers (`kmax1, kmax2, kmax3`) referring to the three spatial directions, to ensure the reciprocal space sum is equally accurate in all directions. The values of `kmax1`, `kmax2` and `kmax3` must be commensurate with the cell geometry to ensure the same minimum wavelength is used in all directions. For a cubic cell set `kmax1` = `kmax2` = `kmax3`. However, for example, in a cell with dimensions $2A = 2B = C$ (ie. a tetragonal cell, longer in the c direction than the a and b directions) use 2`kmax1` = 2`kmax2` = (`kmax3`).

If the values for the `kmax` used are too small, the Ewald sum will produce spurious results. If values that are too large are used, the results will be correct but the calculation will consume unnecessary amounts of *cpu* time. The amount of *cpu* time increases with `kmax1` × `kmax2` × `kmax3`.

3.3.5.2 Hautman Klein Ewald Optimisation

Setting the HKE parameters can also be achieved rather simply, by the use of a **hke precision** directive in the CONTROL file e.g.

hke precision 1d-6 1 1

which specifies the required accuracy of the HKE convergence functions, plus two additional integers; the first specifying the order of the HKE expansion (`nhko`) and the second the

maximum lattice parameter (`nlatt`). DL_POLY_2 will permit values of `nhko` from 1-3, meaning the HKE Taylor series expansion may range from zeroth to third order. Also `nlatt` may range from 1-2, meaning that (1) the nearest neighbour, and (2) and next nearest neighbour, cells are explicitly treated in the real space part of the Ewald sum. Increasing either of these parameters will increase the accuracy, but also substantially increase the cpu time of a simulation. The recommended value for both these parameters is 1 and if *both* these integers are left out, the default values will be adopted.

As with the standard Ewald and SPME methods, the user may set alternative control parameters with the CONTROL file **hke sum** directive e.g.

hke sum 0.05 6 6 1 1

which would set $\alpha = 0.05$ Å^{-1}, `kmax1 = 6`, `kmax2 = 6`. Once again one may check the accuracy by comparing the Coulombic energy with the virial, as described above. The last two integers specify, once again, the values of `nhko` and `nlatt` respectively. (Note it is possible to set either of these to zero in this case.)

Estimating the parameters required for a given simulation follows a similar procedure as for the standard Ewald method (above), but is complicated by the occurrence of higher orders of the convergence functions. Firstly a suitable value for α may be obtained when `nlatt`=0 from the rule: $\alpha = \beta/r_{cut}$, where r_{cut} is the largest real space cutoff compatible with a single MD cell and β=(3.46,4.37,5.01,5.55) when `nhko`=(0,1,2,3) respectively. Thus in the usual case where `nhko`=1, β=4.37. When `nlatt`$\neq$0, this β value is multiplied by a factor $1/(2 * nlatt + 1)$.

The estimation of `kmax1,2` is the same as that for the standard Ewald method above. Note that if any of these parameters prove to be insufficiently accurate, DL_POLY_2 will issue an error in the OUTPUT file, and indicate whether it is the real or reciprocal space sums that is questionable.

3.4 DL_POLY_2 Error Processing

3.4.1 The DL_POLY_2 Internal Error Facility

DL_POLY_2 contains a number of in-built error checks scattered throughout the package which detect a wide range of possible errors. In all cases, when an error is detected the subroutine ERROR is called, resulting in an appropriate message and termination of the program execution (either immediately, or after additional processing.).

Users intending to insert new error checks should ensure that all error checks are performed *concurrently* on *all* nodes, and that in circumstances where a different result may obtain on different nodes, a call to the global status routine GSTATE is made to set the appropriate global error flag on all nodes. Only after this is done, a call to subroutine ERROR may be made. An example of such a procedure might be:

```
logical safe
safe=(test_condition)
```

```
call gstate(safe)
if(.not.safe) call error(node_id,message_number)
```

In this example it is assumed that the logical operation *test_condition* will result in the answer *.true.* if it is safe for the program to proceed, and *.false.* otherwise. The call to ERROR requires the user to state the identity of the calling node (`node_id`), so that only the nominated node in ERROR (i.e. node 0) will print the error message. The variable `message_number` is an integer used to identify the appropriate message to be printed.

In all cases, if ERROR is called with a *non-negative* message number, the program run terminates. If the message number is *negative*, execution continues, but even in this case DL_POLY_2 will terminate the job at a more appropriate place. This feature is used in processing the CONTROL and FIELD file directives. A possible modification users may consider is to dump additional data before the call to ERROR is made.

A full list of the DL_POLY_2 error messages and the appropriate user action can be found in Appendix C of this document.

Chapter 4

DL_POLY_2 Data Files

Scope of Chapter

This chapter describes all the input and output files for DL_POLY_2 , examples of which are to be found in the *data* sub-directory.

4.1 The INPUT files

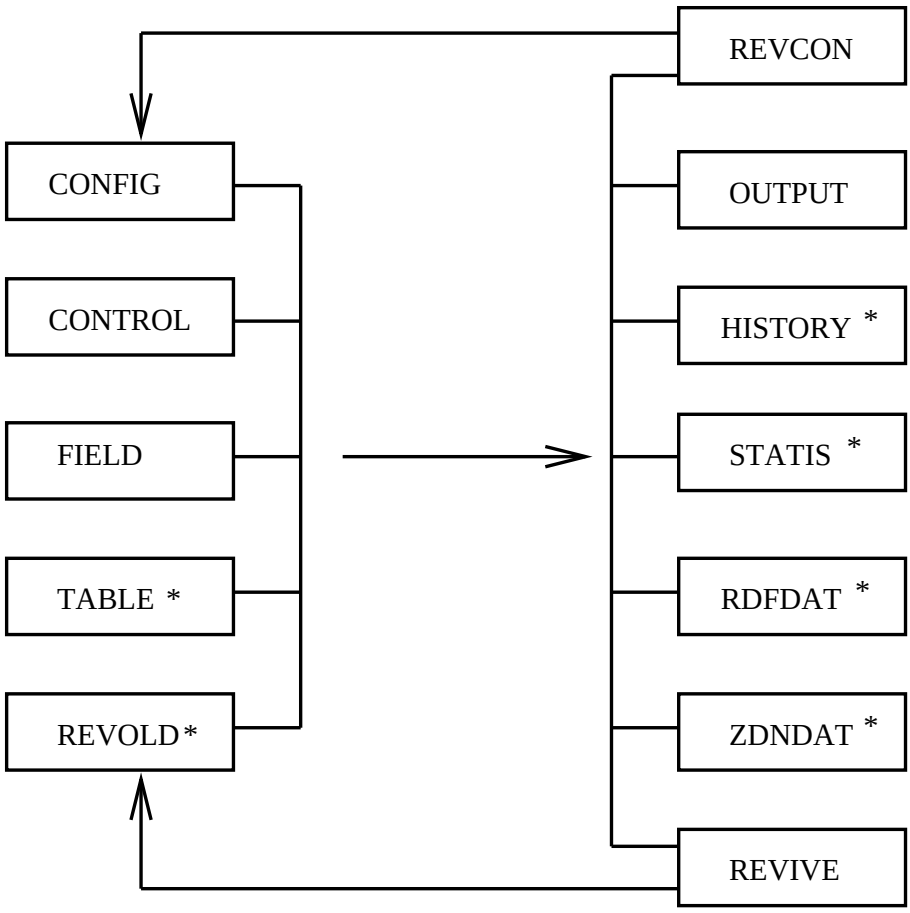


Figure 4.1: DL_POLY_2 input (left) and output (right) files. Note: files marked with an asterisk are non-mandatory.

DL_POLY_2 requires six input files named CONTROL, CONFIG, FIELD, TABLE, TABEAM and REVOLD. The first three files are mandatory, while TABLE and TABEAM are used only to input certain kinds of pair potential, and are not always required. REVOLD is required only if the job represents a continuation of a previous job. In the following sections we describe the form and content of these files.

4.1.1 The CONTROL File

The CONTROL file is read by the subroutine SIMDEF and defines the control variables for running a DL_POLY_2 job. It makes extensive use of **directives** and **keywords**. Directives are character strings that appear as the first entry on a data record (or line) and which

invoke a particular operation or provide numerical parameters. Also associated with each directive may be one or more keywords, which may qualify a particular directive by, for example, adding extra options. Directives have the following general form:

keyword [*options*] {`data`}

The keyword and options are text fields, while the data options are numbers (integers or reals).

Directives can appear in any order in the CONTROL file, except for the **finish** directive which marks the end of the file. Some of the directives are mandatory (for example the **timestep** directive that defines the timestep), others are optional.

This way of constructing the file is very convenient, but it has inherent dangers. It is, for example, quite easy to specify the same directive more than once, or specify contradictory directives, or invoke algorithms that do not work together. By and large DL_POLY_2 tries to sort out these difficulties and print helpful error messages, but it does not claim to be foolproof. Fortunately in most cases the CONTROL file will be small and easy to check visually. It is important to think carefully about a simulation beforehand and ensure that DL_POLY_2 is being asked to do something that is physically reasonable. It should also be remembered that the present capabilites the package may not allow the simulation required and it may be necessary for you yourself to add new features.

An example CONTROL file appears below. The directives and keywords appearing are described in the following section.

```
DL_POLY TEST CASE 1: K Na disilicate glass

temperature      1000.0
pressure         0.0000
ensemble nve

integrator leapfrog
steps               500
equilibration       200
multiple step         5
scale                10
print                10
stack               100
stats                10
rdf                  10

timestep         0.0010
primary cutoff   9.0000
cutoff           12.030
delr width       1.0000
rvdw cutoff      7.6000
```

```
ewald precision  1.0E-5
print rdf

job time              1200.0
close time            100.00

finish
```

4.1.1.1 The CONTROL file format

The file is free-formatted, integers, reals and additional keywords are entered following the keyword on each record. Real and integer numbers must be separated by a non-numeric character (preferably a space or comma) to be correctly interpreted. No logical variables appear in the control file. Comment records (beginning with a #) and blank lines may be added to aid legibility (see example above). The CONTROL file is not case sensitive.

- The first record in the CONTROL file is a header 80 characters long, to aid identification of the file.
- The last record is a **finish** directive, which marks the end of the input data.

Between the header and the **finish** directive, a wide choice of control directives may be inserted. These are described below.

4.1.1.2 The CONTROL File Directives

The directives available are as follows.

directive:	**meaning:**
all pairs	use all pairs for electrostatic calculations
cap *f*	cap forces during equilibration period *f* is maximum cap in units of kT/Å (default *f*=1000)
close time *f*	set job closure time to *f* seconds
collect	include equilibration data in overall statistics
coul	calculate coulombic forces
cut *f*	set required forces cutoff to *f* (Å)
distan	calculate coulombic forces using distance dependent dielectric
delr *f*	set Verlet neighbour list shell width to *f* (Å)
ensemble nve	select NVE ensemble (default)
ensemble nvt ber *f*	select NVT ensemble with Berendsen thermostat with relaxation constant *f* (ps)
ensemble nvt evans	

select NVT ensemble with Evans thermostat

ensemble nvt hoover f
select NVT ensemble with Hoover-Nose thermostat
with relaxation constant f (ps)

ensemble npt ber f_1 f_2
select Berendsen NPT ensemble with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble npt hoover f_1 f_2
select Hoover NPT ensemble with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble nst ber f_1 f_2
select Berendsen N$\underline{\underline{\sigma}}$T ensemble, with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble nst hoover f_1 f_2
select Hoover N$\underline{\underline{\sigma}}$T ensemble with f_1, f_2
as the thermostat and barostat relaxation times (ps)

ensemble pmf select (NVE) potential of mean force ensemble

eps f set relative dielectric constant to f (default 1.0)

equil n equilibrate simulation for first n timesteps

ewald precision f select Ewald sum for electrostatics, with
automatic parameter optimisation ($0 < f < .5$)

ewald sum α *k1 k2 k3*
select Ewald sum for electrostatics, with:
α = Ewald convergence parameter (Å^{-1})
k1 = maximum k-vector index in x-direction
k2 = maximum k-vector index in y-direction
k3 = maximum k-vector index in z-direction

finish close the CONTROL file (last data record)

hke precision f i j select HK-Ewald sum for electrostatics, with
automatic parameter optimisation ($0 < f < .5$)
i = required order of HKE expansion (recommend 1)
j = required lattice sum order (recommend 1)

hke sum α *k1 k2 i j*
select HK-Ewald sum for electrostatics, with:
α = Ewald convergence parameter (Å^{-1})
k1 = maximum g-vector index in x-direction
k2 = maximum g-vector index in y-direction
nhko = required order of HKE expansion (recommend 1)
nlatt = required lattice sum order (recommend 1)

integrator *type* select *type* of integration algorithm:
leapfrog : leapfrog integration algorithm (default)
velocity : velocity Verlet integration algorithm

job time f set job time to f seconds

mult *n*	set multiple timestep (multi-step)interval (activated when *n*>2)
no elec	ignore coulombic interactions
no vdw	ignore short range (non-bonded) interactions
pres *f*	set required system pressure to *f* katm (target pressure for constant pressure ensembles)
prim *f*	set primary cutoff to *f* (Å) (for multiple timestep algorithm only)
print *n*	print system data every *n* timesteps
print rdf	print radial distribution functions
quaternion *f*	set quaternion tolerance to *f* (default 10^{-8})
rdf *f*	calculate radial distribution functions at intervals of *f* timesteps
reaction	select reaction field electrostatics
restart	restart job from end point of previous run (i.e. continue current simulation)
restart scale	restart job from previous run with temperature scaling (i.e. begin a new simulation from older run)
rvdw *f*	set required vdw forces cutoff to *f* (Å)
scale *n*	rescale atomic velocities every *n* steps (during equilibration)
shake *f*	set shake tolerance to *f* (default 10^{-8})
shift	calculate electrostatic forces using shifted coulombic potential
spme precision *f*	select Ewald sum for electrostatics, with automatic parameter optimisation ($0 < f < .5$)
spme sum *α k1 k2 k3*	select Ewald sum for electrostatics, with: α = Ewald convergence parameter (Å^{-1}) *k1* = maximum k-vector index in x-direction *k2* = maximum k-vector index in y-direction *k3* = maximum k-vector index in z-direction
stack *n*	set rolling average stack to *n* timesteps
stats *n*	accumulate statistics data every *n* timesteps
steps *n*	run simulation for *n* timesteps
temp *f*	set required simulation temperature to *f* K
traj *i j k*	write HISTORY file with controls: *i* = start timestep for dumping configurations *j* = timestep interval between configurations *k* = data level (i.e. variable `keytrj` see table 4.3)
timestep *f*	set timestep to *f* ps
zden	calculate the z-density profile
zero	perform zero temperature MD run

4.1.1.3 Further Comments on the CONTROL File

1. A number of the directives (or their **mutually exclusive** alternatives) are **mandatory**:
   (a) **timestep**: specifying the simulation timestep;
   (b) **temp** or **zero** : specifying the system temperature (not mutually exclusive);
   (c) **ewald sum** or **ewald precision** or **spme sum** or **spme precision** or **hke sum** or **hke precision** or **coul** or **shift** or **distan** or **reaction** or **no elec**: specifying the required coulombic forces option;
   (d) **cut** and **delr**: specifying the short range forces cutoff and Verlet strip;
   (e) **prim**: specifying primary forces cutoff (**if mult>2 only**).

2. The **job time** and **close time** directives are required to ensure a controlled close down procedure when a job runs out of time. The time specified by the **job time** directive indicates the total time allowed for the job. (This must obviously be set equal to the time specified to the operating system when the job is submitted.) The **close time** directive represents the time DL_POLY_2 will require to write and close all the data files at the end of processing. This means the *effective* processing time limit is equal to the job time minus the close time. Thus when DL_POLY_2 reaches the effective job time limit it begins the close down procedure with enough time in hand to ensure the files are correctly written. In this way you may be sure the restart files etc. are complete when the job terminates. Note that setting the close time too small will mean the job will crash before the files have been finished. If it is set too large DL_POLY_2 will begin closing down too early. How large the close time needs to be to ensure safe close down is system dependent and a matter of experience. It generally increases with the job size.

3. The starting options for a simulation are governed by the keyword **restart**. If this is **not** specified in the CONTROL file, the simulation will start as new. If specifed, it will either continue a previous simulation (**restart**) or start a new simulation with initial temperature scaling of the previous configuration (**restart scale**) or without initial temperature scaling (**restart noscale**). Internally these options are handled by the integer variable `keyres`, which is explained in table 4.1.

4. The various **ensemble** options (i.e. **nve**, **nvt ber**, **nvt evans, nvt hoover, npt ber, npt hoover**, **nst ber**, **nst hoover**) are mutually exclusive, though none is mandatory (the default is the NVE ensemble). These options are handled internally by the integer variable `keyens`. The meaning of this variable is explained in table 4.2.

5. The force selection directives **ewald sum**, **ewald precision**, **reaction**, **coul**, **shift**, **dist**, **no elec** and **no vdw** are handled internally by the integer variable `keyfce`. See table 4.4 for an explanation of this variable. Note that these options are mutually exclusive.

6. The choice of reaction field electrostatics (directive **reaction**) requires also the specification of the relative dielectric constant external to the cavity. This is specified in the **eps** directive.

7. DL_POLY_2 uses as many as three different potential cutoffs. These are as follows:

   (a) `cut` - this is the universal cutoff. It applies to the real space part of the electrostatics calculations and to the van der Waals potentials if no other cutoff is applied;

   (b) `rvdw` - this is the user-specified cutoff for the van der Waals potentials. If not specified its value defaults to `rcut`. It cannot exceed `cut`;

   (c) `rprim` - this is used in the multiple timestep algorithm to specify the primary atom region (see section 2.5.8). It has no meaning if the multiple timestep option is not used.

8. Some directives are optional. If not specified DL_POLY_2 will take default values if necessary. The defaults appear in the above table.

9. The **zero** directive, enables a zero temperature simulation. This is intended as a crude energy minimizer to help relax a system before a simulation begins. It should not be thought of as a true energy minimization method.

10. The DL_POLY_2 multiple timestep option is invoked if the number appearing with the **mult** directive is greater than 2. This number (stored in the variable `multt`) specifies the number of timesteps (the multi-step) that elapse between partitions of the full Verlet neighbour list into primary and secondary atoms.

11. If a multiple time-step is used, (i.e. `multt`>2), then statistics for radial distribution functions are collected only at updates of the secondary neighbour list. The number specified on the **rdf** directive (the variable `nstbgr`) means that RDF data are accumulated at intervals of `nstbgr`×`multt` timesteps.

12. As a default, DL_POLY_2 does not store statistical data during the equilibration period. If the directive **collect** is used, equilibration data will be incorporated into the overall statistics.

13. The directive **delr** specifies the width of the border to be used in the Verlet neighbour list construction. The width is stored in the variable `delr`. The list is updated whenever two or more atoms have moved a distance of more then `delr`/2 from their positions at the last update of the Verlet list.

Users are advised to study the example CONTROL files appearing in the *data* subdirectory to see how different files are constructed.

Table 4.1: Internal Restart Key

keyres	meaning
0	start new simulation from CONFIG file, and assign velocities from Gaussian distribution.
1	continue current simulation
2	start new simulation from CONFIG file, and rescale velocities to desired temperature
3	start new simulation from CONFIG file, without rescaling the velocities

Table 4.2: Internal Ensemble Key

keyens	meaning
0	Microcanonical ensemble (NVE)
1	Evans NVT ensemble
2	Berendsen NVT ensemble
3	Nosé-Hoover NVT ensemble
4	Berendsen NPT ensemble
5	Nosé-Hoover NPT ensemble
6	Berendsen N$\underline{\underline{\sigma}}$T ensemble
7	Nosé-Hoover N$\underline{\underline{\sigma}}$T ensemble
8	Potential of mean force (NVE) ensemble

Table 4.3: Internal Trajectory File Key

keytrj	meaning
0	coordinates only in file
1	coordinates and velocities in file
2	coordinates, velocities and forces in file

Table 4.4: Non-bonded force key

`keyfce`	meaning
odd	evaluate short-range potentials and electrostatics
even	evaluate Electrostatic potential only
Electrostatics are evaluated as follows:	
0†, 1‡	Ignore Electrostatic interactions
2, 3	Ewald summation
4, 5	distance dependent dielectric constant
6, 7	standard truncated Coulombic potential
8, 9	truncated and shifted Coulombic potential
10,11	Reaction Field electrostatics
12,13	SPME electrostatics
14,15	Hautman-Klein Ewald electrostatics

† `keyfce` = 0 means no non-bonded terms are evaluated.
‡ `keyfce` = 1 means only short-range potentials are evaluated.

4.1.2 The CONFIG File

The CONFIG file contains the dimensions of the unit cell, the key for periodic boundary conditions and the atomic labels, coordinates, velocities and forces. This file is read by the subroutine SYSGEN. (It is also read by the subroutine SIMDEF if the **ewald precision** directive is used.) The first few records of a typical CONFIG file are shown below:

```
IceI structure 6x6x6 unit cells with proton disorder
         2         3
  26.988000000000000   0.000000000000000   0.000000000000000
 -13.494000000000000  23.372293600000000   0.000000000000000
   0.000000000000000   0.000000000000000  44.028000000000000
      OW         1
    -2.505228382         -1.484234330         -7.274585343
    0.5446573999         -1.872177437        -0.7702718106
     3515.939287          13070.74357          4432.030587
      HW         2
    -1.622622646         -1.972916834         -7.340573742
     1.507099154         -1.577400769          4.328786484
     7455.527553         -4806.880540         -1255.814536
      HW         3
    -3.258494716         -2.125627191         -7.491549620
     2.413871957         -4.336956694          2.951142896
    -7896.278327         -8318.045939         -2379.766752
      OW         4
    0.9720599243E-01     -2.503798635         -3.732081894
     1.787340483         -1.021777575         0.5473436377
     9226.455153          9445.662860          5365.202509

etc.
```

4.1.2.1 Format

The file is fixed-formatted: integers as "i10", reals as "f20.0". The header record is formatted as 80 alphanumeric characters.

4.1.2.2 Definitions of Variables

record 1		
`header`	a80	title line
record 2		
`levcfg`	integer	CONFIG file key. See table 4.5 for permitted values
`imcon`	integer	Periodic boundary key. See table 4.6 for permitted values
record 3	omitted if `imcon` = 0	
`cell(1)`	real	x component of a cell vector

`cell(2)`	real	y component of a cell vector
`cell(3)`	real	z component of a cell vector
record 4	omitted if `imcon` = 0	
`cell(4)`	real	x component of b cell vector
`cell(5)`	real	y component of b cell vector
`cell(6)`	real	z component of b cell vector
record 5	omitted if `imcon` = 0	
`cell(7)`	real	x component of c cell vector
`cell(8)`	real	y component of c cell vector
`cell(9)`	real	z component of c cell vector

Subsequent records consists of blocks of between 2 and 4 records depending on the value of the `levcfg` variable. Each block refers to one atom. The atoms must be listed sequentially in order of increasing index. Within each block the data are as follows:

record i		
`atmnam`	a8	atom name.
`index`	integer	atom index
`atmnum`	integer	atomic number
record ii		
`xxx`	real	x coordinate
`yyy`	real	y coordinate
`zzz`	real	z coordinate
record iii	included only if `levcfg` > 0	
`vxx`	real	x component of velocity
`vyy`	real	y component of velocity
`vzz`	real	x component of velocity
record iv	included only if `levcfg` > 1	
`fxx`	real	x component of force
`fyy`	real	y component of force
`fzz`	real	z component of force

Note that on **record i** only the atom name is mandatory, any other items are not read by DL_POLY_2 but may be added to aid alternative uses of the file, for example the DL_POLY_2 Graphical User Interface [8].

4.1.2.3 Further Comments

The CONFIG file has the same format as the output file REVCON (section 4.2.3). When restarting from a previous run of DL_POLY_2 (i.e. using the **restart** or **restart scale** directives in the CONTROL file - above), the CONFIG file must be replaced by the REVCON file, which is renamed as the CONFIG file. The *copy* macro in the *execute* sub-directory of DL_POLY_2 does this for you.

Table 4.5: CONFIG file key (record 2)

levcfg	meaning
0	Coordinates included in file
1	Coordinates and velocities included in file
2	Coordinates, velocities and forces included in file

Table 4.6: Periodic boundary key (record 2)

imcon	meaning
0	no periodic boundaries
1	cubic boundary conditions
2	orthorhombic boundary conditions
3	parallelepiped boundary conditions
4	truncated octahedral boundary conditions
5	rhombic dodecahedral boundary conditions
6	x-y parallelogram boundary conditions with no periodicity in the z direction
7	hexagonal prism boundary conditions

4.1.3 The FIELD File

The FIELD file contains the force field information defining the nature of the molecular forces. It is read by the subroutine SYSDEF. Excerpts from a force field file are shown below. The example is the antibiotic Valinomycin in a cluster of 146 water molecules.

```
Valinomycin Molecule with 146 SPC Waters
UNITS kcal

MOLECULES    2
Valinomycin
NUMMOLS 1
ATOMS 168
    O         16.0000      -0.4160    1
    OS        16.0000      -0.4550    1
    "           "            "        "
    "           "            "        "
    HC         1.0080       0.0580    1
    C         12.0100       0.4770    1
BONDS 78
harm   31   19 674.000      1.44900
harm   33   31 620.000      1.52600
    "    "    "    "          "
    "    "    "    "          "
harm  168   19 980.000      1.33500
harm  168  162 634.000      1.52200
CONSTRAINTS 90
   20   19    1.000017
   22   21    1.000032
    "    "      "
    "    "      "
  166  164    1.000087
  167  164    0.999968
ANGLES 312
harm   43    2   44  200.00      116.40
harm   69    5   70  200.00      116.40
  "    "    "    "     "            "
  "    "    "    "     "            "
harm   18  168  162  160.00      120.40
harm   19  168  162  140.00      116.60
DIHEDRALS 371
harm    1   43    2   44  2.3000      180.00
harm   31   43    2   44  2.3000      180.00
  "    "    "    "    "     "           "
  "    "    "    "    "     "           "
```

```
cos   149   17  161   16  10.500      180.00
cos   162   19  168   18  10.500      180.00
FINISH
SPC Water
NUMMOLS 146
ATOMS  3
    OW       16.0000     -0.8200
    HW        1.0080      0.4100
    HW        1.0080      0.4100
CONSTRAINTS  3
    1    2    1.0000
    1    3    1.0000
    2    3    1.63299
FINISH
VDW   45
C       C          lj   0.12000      3.2963
C       CT         lj   0.08485      3.2518
"       "          "     "            "
"       "          "     "            "
"       "          "     "            "
OW      OS         lj   0.15100      3.0451
OS      OS         lj   0.15000      2.9400
CLOSE
```

4.1.3.1 Format

The FIELD file is free formatted (though it should be noted that atom names are limited to 8 characters and potential function keys are a maximum of 4 characters). The contents of the file are variable and are defined by the use of **directives**. Additional information is associated with the directives. The file is not case sensitive.

4.1.3.2 Definitions of Variables

The file divides into three sections: general information, molecular descriptions, and non-bonded interaction descriptions, appearing in that order in the file.

4.1.3.2.1 General information

The first record in the FIELD file is the title. It must be followed by the **units** directive. Both of these are mandatory. These records may optionally be followed by the **neut** directive.

record 1

header	a80	field file header

record 2

units	a40	Unit of energy used for input and output
record 3 (optional)		
neut	a40	activate the neutral/charge groups option for the electrostatic calculations

The energy units on the **units** directive are described by additional keywords:

a eV, for electron-volts

b kcal, for k-calories mol^{-1}

c kJ, for k-Joules mol^{-1}

d K, for $Kelvin^{-1}$

e internal, for DL_POLY_2 internal units (10 J mol^{-1}).

If no units keyword is entered, DL_POLY_2 units are assumed for both input and output. The units keyword may appear anywhere on the data record provided it does not exceed column 40. The **units** directive only affects the input and output interfaces, all internal calculations are handled using DL_POLY_2 units.

4.1.3.2.2 Molecular details

It is important for the user to understand that there is an organisational correspondence between the FIELD file and the CONFIG file described above. It is required that the order of specification of molecular types and their atomic constituents in the FIELD file follows the order in which they appear in the CONFIG file. Failure to adhere to this common sequence will be detected by DL_POLY_2 and result in premature termination of the job. It is therefore essential to work from the CONFIG file when constructing the FIELD file. It is not as difficult as it sounds!

The entry of the molecular details begins with the mandatory directive:

molecules *n*

where n is an integer specifying the number of different *types* of molecule appearing in the FIELD file. Once this directive has been encountered, DL_POLY_2 enters the *molecular description* environment in which only molecular decription keywords and data are valid.

Immediately following the **molecules** directive, are the records defining individual molecules:

1. *name-of-molecule*
   which can be any character string up to 80 characters in length. (Note: this is not a directive, just a simple character string.)

2. **nummols *n***
   where n is the number of times a molecule of this type appears in the simulated system. The molecular data then follow in subsequent records:

3. **atoms *n***
   where n indicates the number of atoms in this type of molecule. A number of records follow, each giving details of the atoms in the molecule i.e. site names, masses and charges. Each record carries the entries:

`sitnam`	a8	atomic site name
`weight`	real	atomic site mass
`chge`	real	atomic site charge
`nrept`	integer	repeat counter
`ifrz`	integer	'frozen' atom (if `ifrz`> 0)
`igrp`	integer	neutral/charge group number

   Note that these entries are order sensitive. Do not leave blank entries unless all parameters appearing after the last specified are void. The integer `nrept` need not be specified (in which case a value of 1 is assumed.) A number greater than 1 specified here indicates that the next (`nrept` - 1) entries in the CONFIG file are ascribed the atomic characteristics given in the current record. The sum of the repeat numbers for all atoms in a molecule should equal the number specified by the **atoms** directive.

4. **shell *n m***
   where n is the number of core-shell units and m is an integer specifying which shell model is required:

   - m=1 for adiabatic shell model;
   - m=2 for relaxed shell model;

   Each of the subsequent n records contains:

index 1	integer	site index of core
index 2	integer	site index of shell
spring	real	force constant of core-shell spring

   The spring force constant is entered in units of `engunit` Å^{-2}, where `engunit` is the energy unit specified in the **units** directive. The adiabatic and relaxed shell models are mutually exclusive options in the same simulation.

   Note that the atomic site indices referred to in this table are indices arising from numbering each atom in the molecule from 1 to the number specified in the **atoms**

directive for this molecule. This same numbering scheme should be used for all descriptions of this molecule, including the **bonds**, **constraints**, **angles**, and **dihedrals** entries described below. DL_POLY_2 will itself construct the global indices for all atoms in the systems.

This directive (and associated data records) need not be specified if the molecule contains no core-shell units.

5. **bonds *n***
where n is the number of flexible chemical bonds in the molecule. Each of the subsequent n records contains:

bond key	a4	see table 4.7
index 1	integer	first atomic site in bond
index 2	integer	second atomic site in bond
variable 1	real	potential parameter see table 4.7
variable 2	real	potential parameter see table 4.7
variable 3	real	potential parameter see table 4.7
variable 4	real	potential parameter see table 4.7

The meaning of these variables is given in table 4.7. This directive (and associated data records) need not be specified if the molecule contains no flexible chemical bonds. See the note on the atomic indices appearing under the **shell** directive above.

6. **constraints *n***
where n is the number of constraint bonds in the molecule. Each of the following n records contains:

index 1	integer	first atomic index
index 2	integer	second atomic index
bondlength	real	constraint bond length

This directive (and associated data records) need not be specified if the molecule contains no constraint bonds. See the note on the atomic indices appearing under the **shell** directive above.

7. **pmf *b***
where b is the potential of mean force bondlength (Å). There follows the definitions of two PMF units:

   (a) **pmf unit** n_1
   where n_1 is the number of sites in the first unit. The subsequent n_1 records provide the site indices and weighting. Each record contains:

Table 4.7: Chemical bond potentials

key	potential type	Variables (1-4)				functional form
harm -hrm	Harmonic	k	r_0			$U(r) = \frac{1}{2}k(r-r_0)^2$
mors -mrs	Morse	E_0	r_0	k		$U(r) = E_0[\{1-\exp(-k(r-r_0))\}^2 - 1]$
12-6 -126	12-6	A	B			$U(r) = \left(\frac{A}{r^{12}}\right) - \left(\frac{B}{r^6}\right)$
rhrm -rhm	Restraint	k	r_0	r_c		$U(r) = \frac{1}{2}k(r-r_0)^2 \quad \lvert r-r_0\rvert \le r_c$ $U(r) = \frac{1}{2}kr_c^2 + kr_c(\lvert r-r_0\rvert - r_c) \quad \lvert r-r_0\rvert > r_c$
quar -qur	Quartic	k	r_0	k'	k''	$U(r) = \frac{k}{2}(r-r_0)^2 + \frac{k'}{3}(r-r_0)^3 + \frac{k''}{4}(r-r_0)^4$
buck -bck	Buckingham	A	ρ	C		$U(r) = Aexp(-r/\rho) - C/r^6$

Note: bond potentials with a dash (-) as the first character of the keyword, do not contribute to the *excluded atoms list* (see section 2.1). In this case DL_POLY_2 will also calculate the nonbonded pair potentials between the described atoms, unless these are deactivated by another potential specification.

index	integer	atomic site index
weight	real	site weighting

(b) **pmf unit** n_2
where n_2 is the number of sites in the second unit. The subsequent n_2 records provide the site indices and weighting. Each record contains:

index	integer	atomic site index
weight	real	site weighting

This directive (and associated data records) need not be specified if no PMF constraints are present. See the note on the atomic indices appearing under the **shell** directive above. The pmf bondlength applies to the distance between the centres of

the two pmf units. The centre, $\underline{R}$, of each unit is given by

$$\underline{R} = \frac{\sum_\alpha w_\alpha \underline{r}_\alpha}{\sum_\alpha w_\alpha}$$

where $\underline{r}_\alpha$ is a site position and w_α the site weighting. Note that the pmf constraint is intramolecular. To define a constraint between two molecules, the molecules must be described as part of the same DL_POLY "molecule". This is illustrated in test case 6, where a pmf constraint is imposed between a potassium ion and the centre of mass of a water molecule. DL_POLY_2 allows only one type of pmf constraint per system. The value of `nummols` for this molecule determines the number of pmf constraint in the system.

Note that the directive **ensemble pmf** must be specified in the CONTROL file for this option to be implemented correctly.

8. **angles *n***
where n is the number of valence angle bonds in the molecule. Each of the n records following contains:

angle key	a4	potential key. See table 4.8
index 1	integer	first atomic index
index 2	integer	second atomic index (central site)
index 3	integer	third atomic index
variable 1	real	potential parameter see table 4.8
variable 2	real	potential parameter see table 4.8

The meaning of these variables is given in table 4.8. See the note on the atomic indices appearing under the **shell** directive above. This directive (and associated data records) need not be specified if the molecule contains no angular terms.

Table 4.8: Valence Angle potentials

key	potential type	Variables (1-4)				functional form†
harm -hrm	Harmonic	k	θ_0			$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2$
quar -qur	Quartic	k	θ_0	k'	k''	$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 + \frac{k'}{3}(\theta - \theta_0)^3 + \frac{k''}{4}(\theta - \theta_0)^4$
thrm -thm	Truncated harmonic	k	θ_0	ρ		$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
shrm -shm	Screened harmonic	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs1 -bv1	Screened Vessal[24]	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{8(\theta-\theta_0)^2}\left\{[(\theta_0 - \pi)^2 - (\theta - \pi)^2]^2\right\}$ $\exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs2 -bv2	Truncated Vessal[25]	k	θ_0	a	ρ	$U(\theta) = k[\theta^a(\theta - \theta_0)^2(\theta + \theta_0 - 2\pi)^2 - \frac{a}{2}\pi^{a-1}$ $(\theta - \theta_0)^2(\pi - \theta_0)^3] \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
hcos -hcs	Harmonic Cosine	k	θ_0			$U(\theta) = \frac{k}{2}(cos(\theta) - cos(\theta_0))^2$
cos -cos	Cosine	A	δ	m		$U(\theta) = A[1 + cos(m\theta - \delta)]$
mmsb -msb	MM Stretch-bend	A	θ_0	d_{ab}	d_{ac}	$U(\theta) = A(\theta - \theta_0)(r_{ab} - d_{ab})(r_{ac} - d_{ac})$

†θ is the a-b-c angle.

Note: valence angle potentials with a dash (-) as the first character of the keyword, do not contribute to the *excluded atoms list* (see section 2.1). In this case DL_POLY_2 will calculate the nonbonded pair potentials between the described atoms.

9. **dihedrals *n***
where n is the number of dihedral interactions present in the molecule. Each of the following n records contains:

dihedral key	a4	potential key. See table 4.9
index 1	integer	first atomic index
index 2	integer	second atomic index
index 3	integer	third atomic index
index 4	integer	fourth atomic index
variable 1	real	potential parameter see table 4.9
variable 2	real	potential parameter see table 4.9
variable 3	real	potential parameter see table 4.9
variable 4	real	1-4 electrostatic interaction scale factor.
variable 5	real	1-4 Van der Waals interaction scale factor.

The meaning of the variables 1-3 is given in table 4.9. The variables 4 and 5 specify the scaling factor for the 1-4 electrostatic and Van der Waals nonbonded interactions respectively. This directive (and associated data records) need not be specified if the molecule contains no dihedral angle terms. See the note on the atomic indices appearing under the **shell** directive above.

10. **inversions *n***
where n is the number of inversion interactions present in the molecule. Each of the following n records contains:

inversion key	a4	potential key. See table 4.10
index 1	integer	first atomic index
index 2	integer	second atomic index
index 3	integer	third atomic index
index 4	integer	fourth atomic index
variable 1	real	potential parameter see table 4.10
variable 2	real	potential parameter see table 4.10

The meaning of the variables 1-2 is given in table 4.10. This directive (and associated data records) need not be specified if the molecule contains no inversion angle terms. See the note on the atomic indices appearing under the **shell** directive above.

11. **rigid *n***
where n is the number of rigid units in the molecule. It is followed by at least n records, each specifying the sites in a rigid unit:

m	integer	number of sites in rigid unit
site 1	integer	first site atomic index

Table 4.9: Dihedral Angle Potentials

key	potential type	Variables (1-4)				functional form‡
cos	Cosine	A	δ	m		$U(\phi) = A\,[1 + \cos(m\phi - \delta)]$
harm	Harmonic	k	ϕ_0			$U(\phi) = \frac{1}{2}k(\phi - \phi_0)^2$
hcos	Harmonic cosine	k	ϕ_0			$U(\phi) = \frac{k}{2}(cos(\phi) - cos(\phi_0))^2$
cos3	Triple cosine	A_1	A_2	A_3		$U(\phi) = \frac{1}{2}A_1(1 + cos(\phi)) + \frac{1}{2}A_2(1 - cos(2\phi))$ $+\frac{1}{2}A_3(1 + cos(3\phi))$
ryck	Ryckaert-Bellemans	A				$U(\phi) = A(a_0 + a_1 cos\phi - a_2 cos^2\phi + a_3 cos^3\phi +$ $a_4 cos^4\phi + a_5 cos^5\phi)$ $[a_0 \rightarrow a_5$ pre-set]
rbf	Fluorinated Ryckaert-Bellemans	B				$U(\phi) = B(b_0 - b_1 cos\phi - b_2 cos^2\phi - b_3 cos^3\phi +$ $b_4 cos^4\phi + b_5 cos^5\phi + b_6 exp(-b_7(\phi - \pi))$ $[b_0 \rightarrow b_6$ pre-set]
opls	OPLS	A_0	A_1	A_2	A_3	$U(\phi) = A_0 + \frac{1}{2}(A_1(1 + cos(\phi)) + A_2(1 - cos(2\phi))$ $+A_3(1 + cos(3\phi)))$

‡ϕ is the a-b-c-d dihedral angle.

site 2	integer	second site atomic index
site 3	integer	third site atomic index
..	..	*etc.*
site m	integer	m'th site atomic index

Up to 15 sites can be specified on the first record. Additional records are used if necessary. Up to 16 sites are specified per record thereafter.

This directive (and associated data records) need not be specified if the molecule contains no rigid units. See the note on the atomic indices appearing under the **shell** directive above.

12. **teth *n***
where n is the number of tethered atoms in the molecule. It is followed by n records specifying the tethered sites in the molecule:

tether key	a4	tethering potential key see table 4.11
index	integer	atomic index

Table 4.10: Inversion Angle Potentials

key	potential type	Variables (1-2)		functional form‡
harm	Harmonic	k	ϕ_0	$U(\phi) = \frac{1}{2}k(\phi - \phi_0)^2$
hcos	Harmonic cosine	k	ϕ_0	$U(\phi) = \frac{k}{2}(cos(\phi) - cos(\phi_0))^2$
plan	Planar	A		$U(\phi) = A\,[1 - \cos(\phi)]$

‡ϕ is the inversion angle.

variable 1	real	potential parameter see table 4.11
variable 2	real	potential parameter see table 4.11
variable 3	real	potential parameter see table 4.11
variable 4	real	potential parameter see table 4.11

This directive (and associated data records) need not be specified if the molecule contains no tethered atoms. See the note on the atomic indices appearing under the **shell** directive above.

Table 4.11: Tethering potentials

key	potential type	Variables (1-3)			functional form
harm	Harmonic	k			$U(r) = \frac{1}{2}kr^2$
rhrm	Restraint	k	r_c		$U(r) = \frac{1}{2}kr^2 \qquad r \le r_c$ $U = \frac{1}{2}kr_c^2 + kr_c(r - r_c) \qquad r > r_c$
quar	Quartic	k	k'	k''	$U(r) = \frac{k}{2}r^2 + \frac{k'}{3}r^3 + \frac{k''}{4}r^4$

13. **finish**
This directive is entered to signal to DL_POLY_2 that the entry of the details of a molecule has been completed.

The entries for a second molecule may now be entered, beginning with the *name-of-molecule* record and ending with the **finish** directive.

The cycle is repeated until all the types of molecules indicated by the **molecules** directive have been entered.

The user is recommended to look at the example FIELD files in the *data* directory to see how typical FIELD files are constructed.

4.1.3.3 Non-bonded Interactions

Non-bonded interactions are identified by atom types as opposed to specific atomic indices. The first type of non-bonded potentials are the pair potentials. The input of pair potential data is signalled by the directive:

vdw *n*

where n is the number of pair potentials to be entered. There follows n records, each specifying a particular pair potential in the following manner:

atmnam 1	a8	first atom type
atmnam 2	a8	second atom type
key	a4	potential key. See table 4.12
variable 1	real	potential parameter see table 4.12
variable 2	real	potential parameter see table 4.12
variable 3	real	potential parameter see table 4.12
variable 4	real	potential parameter see table 4.12
variable 5	real	potential parameter see table 4.12

The variables pertaining to each potential are described in table 4.12. Note that any pair potential not specified in the FIELD file, will be assumed to be zero.

The specification of three body potentials is initiated by the directive:

tbp *n*

where n is the number of three-body potentials to be entered. There follows n records, each specifying a particular three body potential in the following manner:

atmnam 1	a8	first atom type
atmnam 2	a8	second atom type (central site)
atmnam 3	a8	third atom type
key	a4	potential key. See table 4.13

Table 4.12: Definition of pair potential functions and variables

key	potential type	Variables (1-5)					functional form
12-6	12-6	A	B				$U(r) = \left(\frac{A}{r^{12}}\right) - \left(\frac{B}{r^6}\right)$
lj	Lennard-Jones	ϵ	σ				$U(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$
nm	n-m	E_o	n	m	r_0		$U(r) = \frac{E_o}{(n-m)}\left[m\left(\frac{r_o}{r}\right)^n - n\left(\frac{r_o}{r}\right)^m\right]$
buck	Buckingham	A	ρ	C			$U(r) = A\ \exp\left(-\frac{r}{\rho}\right) - \frac{C}{r^6}$
bhm	Born-Huggins -Meyer	A	B	σ	C	D	$U(r) = A\ \exp[B(\sigma - r)] - \frac{C}{r^6} - \frac{D}{r^8}$
hbnd	12-10 H-bond	A	B				$U(r) = \left(\frac{A}{r^{12}}\right) - \left(\frac{B}{r^{10}}\right)$
snm	Shifted force† n-m [27]	E_o	n	m	r_0	r_c‡	$U(r) = \frac{\alpha E_o}{(n-m)}\times$ $\left[m\beta^n\left\{\left(\frac{r_o}{r}\right)^n - \left(\frac{1}{\gamma}\right)^n\right\} - n\beta^m\left\{\left(\frac{r_o}{r}\right)^m - \left(\frac{1}{\gamma}\right)^m\right\}\right]$ $+\frac{nm\alpha E_o}{(n-m)}\left(\frac{r-\gamma r_o}{\gamma r_o}\right)\left\{\left(\frac{\beta}{\gamma}\right)^n - \left(\frac{\beta}{\gamma}\right)^m\right\}$
mors	Morse	E_0	r_0	k			$U(r) = E_0[\{1 - \exp(-k(r - r_0))\}^2 - 1]$
tab	Tabulation						tabulated potential

† Note: in this formula the terms α, β and γ are compound expressions involving the variables E_o, n, m, r_0 and r_c. See section 2.3.1 for further details.
‡ Note: r_c defaults to the general van der Waals cutoff (`rvdw` or `rcut`) if it is set to zero or not specified or not specified in the FIELD file.

Table 4.13: Three-body potentials

key	potential type	Variables (1-4)				functional form†
thrm	Truncated harmonic	k	θ_0	ρ		$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
shrm	Screened harmonic	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs1	Screened Vessal[24]	k	θ_0	ρ_1	ρ_2	$U(\theta) = \frac{k}{8(\theta-\theta_0)^2}\left\{[(\theta_0 - \pi)^2 - (\theta - \pi)^2]^2\right\}$ $\exp[-(r_{ij}/\rho_1 + r_{ik}/\rho_2)]$
bvs2	Truncated Vessal[25]	k	θ_0	a	ρ	$U(\theta) = k[\theta^a(\theta - \theta_0)^2(\theta + \theta_0 - 2\pi)^2 - \frac{a}{2}\pi^{a-1}$ $(\theta - \theta_0)^2(\pi - \theta_0)^3] \exp[-(r_{ij}^8 + r_{ik}^8)/\rho^8]$
hbnd	H-bond [6]	D_{hb}	R_{hb}			$U(\theta) = D_{hb}cos^4(\theta)[5(R_{hb}/r_{jk})^{12} - 6(R_{hb}/r_{jk})^{10}]$

†θ is the a-b-c angle.

variable 1	real	potential parameter see table 4.13
variable 2	real	potential parameter see table 4.13
variable 3	real	potential parameter see table 4.13
variable 4	real	potential parameter see table 4.13
variable 5	real	cutoff range for this potential (Å)

The variables pertaining to each potential are described in table 4.13. Note that the fifth variable is the range at which the three body potential is truncated. The distance is in Å, measured from the central atom.

The specification of four body potentials is initiated by the directive:

fbp *n*

where n is the number of four-body potentials to be entered. There follows n records, each specifying a particular four-body potential in the following manner:

atmnam 1	a8	first atom type (central site)
atmnam 2	a8	second atom type
atmnam 3	a8	third atom type
atmnam 4	a8	fourth atom type
key	a4	potential key. See table 4.14
variable 1	real	potential parameter see table 4.14
variable 2	real	potential parameter see table 4.14
variable 3	real	cutoff range for this potential (Å)

The variables pertaining to each potential are described in table 4.14. Note that the third variable is the range at which the four-body potential is truncated. The distance is in Å, measured from the central atom.

Table 4.14: Four-body Potentials

key	potential type	Variables (1-2)		functional form‡
harm	Harmonic	k	ϕ_0	$U(\phi) = \frac{1}{2}k(\phi - \phi_0)^2$
hcos	Harmonic cosine	k	ϕ_0	$U(\phi) = \frac{k}{2}(cos(\phi) - cos(\phi_0))^2$
plan	Planar	A		$U(\phi) = A\,[1 - \cos(\phi)]$

‡ϕ is the inversion angle.

4.1.3.4 Metal Potentials

Metal potentials in DL_POLY_2 are based on the embedded atom model (EAM) [29, 30] and the Finnis-Sinclair model (FSM)[31] .

The EAM potentials are tabulated and are supplied to DL_POLY_2 in the input file TABEAM4.1.6 (see below).

The FSM potentials are analytical and DL_POLY_2 supports the explicit forms due to: Finnis and Sinclair [31] ; Sutton and Chen [3, 33] ; and Gupta [35] ;

Metal potentials, like van der Waals potentials are also non-bonded potentials and are characterised by atom types rather than specific atomic indices. The input of metal potential data is signalled by the directive:

metal *n*

where n is the number of metal potentials to be entered. There follows n records, each specifying a particular metal potential in the following manner:

atmnam 1	a8	first atom type
atmnam 2	a8	second atom type
key	a4	potential key. See table 4.15
variable 1	real	potential parameter see table 4.15
variable 2	real	potential parameter see table 4.15
variable 3	real	potential parameter see table 4.15
variable 4	real	potential parameter see table 4.15

variable 5	real	potential parameter see table 4.15
variable 6	real	potential parameter see table 4.15
variable 7	real	potential parameter see table 4.15

The variables pertaining to each potential are described in table 4.15. Note that any metal potential not specified in the FIELD file, will be assumed to be zero. This includes cross terms for alloys!

Table 4.15: Metal Potential

key	potential type	Variables (1-7)							functional form
eam	EAM								tabulated potential
fnsc	Finnis-Sinclair	c_0	c_1	c_2	c	A	d	β	$U_i(r) = \frac{1}{2}\sum_{j\neq i}(r_{ij}-c)^2(c_0+c_1 r_{ij}+c_2 r_{ij}^2) - A\sqrt{\rho_i}$ $\rho_i = \sum_{j\neq i}\left[(r_{ij}-c)^2 + \beta\frac{(r_{ij}-d)^3}{d}\right]$
stch	Sutton-Chen	ϵ	a	n	m	c			$U_i(r) = \epsilon\left[\frac{1}{2}\sum_{j\neq i}\left(\frac{a}{r_{ij}}\right)^n - c\sqrt{\rho_i}\right]$ $\rho_i = \sum_{j\neq i}\left(\frac{a}{r_{ij}}\right)^m$
gupt	Gupta	A	r_0	p	B	q_{ij}			$U_i(r) = \frac{1}{2}\sum_{j\neq i} A\exp\left(-p\frac{r_{ij}-r_0}{r_0}\right) - B\sqrt{\rho_i}$ $\rho_i = \sum_{j\neq i}\exp\left(-2q_{ij}\frac{r_{ij}-r_0}{r_0}\right)$

Both EAM and FSM potentials can handle alloys, but care must be taken to enter the cross terms of the potentials explicitly. **Note** that the rules for defining cross terms of the potential are not the usual rules encountered in Lennard-Jones systems see section (2.3.5).

4.1.3.5 The Tersoff Potential

The Tersoff potential [4] is designed to reproduce the effects of covalency in systems composed of group 4 elements in the periodic table (carbon, silicon, germanium etc) and their alloys. Like the metal potentials these are also non-bonded potentials characterised by atom types rather than specific atomic indices. The input of Tersoff potential data is signalled by the directive:

tersoff ***n***

Where n is the number of specified Tersoff potentials. It is followed by $2n$ records specifying n particular Tersoff single atom type parameters and $n(n+1)/2$ records specifying cross atom type parameters in the following manner:

potential 1 : record 1		
atmnam	a8	atom type
key	a4	potential key, see Table 4.16
variable 1	real	potential parameter, see Table 4.16
variable 2	real	potential parameter, see Table 4.16
variable 3	real	potential parameter, see Table 4.16
variable 4	real	potential parameter, see Table 4.16
variable 5	real	cutoff range for this potential (Å) 4.16
potential 1 : record 2		
variable 6	real	potential parameter, see Table 4.16
variable 7	real	potential parameter, see Table 4.16
variable 8	real	potential parameter, see Table 4.16
variable 9	real	potential parameter, see Table 4.16
variable 10	real	potential parameter, see Table 4.16
variable 11	real	potential parameter, see Table 4.16
...	...	...
...	...	...
potential n **: record** $2n-1$		
...	...	...
potential n **: record** $2n$		
...	...	...
cross term 1 **: record** $2n+1$		
atmnam 1	a8	first atom type
atmnam 2	a8	second atom type
variable a	real	potential parameter, see Table 4.16
variable b	real	potential parameter, see Table 4.16
...	...	...
...	...	...
cross term $n(n+1)/2$ **: record** $2n+n(n+1)/2$		
...	...	...

The variables pertaining to each potential are described in Table 4.16.

Note that the fifth variable is the range at which the particular Tersoff potential is truncated. The distance is in Å.

4.1.3.6 External Field

The presence of an external field is flagged by the **extern** directive. The next line in the FIELD file should have another directive indicating what type of field is to be applied. On

Table 4.16: Tersoff Potential

key	potential type	Variables (1-5,6-11,a-b)						functional form
ters	Tersoff (single) (cross)	A S χ	a β ω	B η	b c	R d	 h	Potential form as shown in Section 2.3.3

the following lines comes the `mxfld` parameters, five per line, that describe the field. In the include files supplied with DL_POLY_2 `mxfld` is set to 10.
The variables pertaining to each potential are described in table 4.17.

Table 4.17: External fields

key	potential type	Variables (1-4)				functional form†
elec	Electric field	E_x	E_y	E_z		$\underline{F} = q.\underline{E}$
oshm	Oscillating Shear	A	n			$\underline{F}_x = Acos(2n\pi.z/L_z)$
shrx	Continuous Shear	A	z_0			$\mid z \mid > z_0$: $\underline{v}_x = (1/2)A(\mid z \mid /z)$
grav	Gravitational Field	G_x	G_y	G_z		$\underline{F} = m.\underline{G}$
magn	Magnetic Field	H_x	H_y	H_z		$\underline{F} = q(\underline{v} \times \underline{H})$
sphr	Containing Sphere	A	R_0	n	$R_{\rm cut}$	$r > R_{\rm cut}$: $\underline{F} = A(R_0 - r)^{-n}$
zbnd	Repulsive wall (harmonic)	A	Z_0	$f = \pm 1$		$zf > Z_0 f$: $F_z = -A(z - Z_0)$

4.1.3.7 Closing the FIELD File

The FIELD file must be closed with the directive:

close

which signals the end of the force field data. Without this directive DL_POLY_2 will abort.

4.1.4 The REVOLD File

This file contains statistics arrays from a previous job. It is not required if the current job is not a continuation of a previous run (ie. if the **restart** directive is not present in the CONTROL file - see above). The file is unformatted and therefore not readable by normal people. DL_POLY_2 normally produces the file REVIVE (see section 4.2.4) at the end of a job which contains the statistics data. REVIVE should be copied to REVOLD before a continuation run commences. This may be done by the *copy* macro supplied in the *execute* sub-directory of DL_POLY_2 .

4.1.4.1 Format

The REVOLD file is unformatted. All variables appearing are written in native real*8 representation. Nominally integer quantities (e.g. the timestep number `nstep`) are represented by the the nearest real number. The contents are as follows (the dimensions of array variables are given in brackets and are defined in the appropriate Fortran modules).

record 1:

`nstep`	timestep of final configuration
`numacc`	number of configurations used in averages
`numrdf`	number of configurations used in rdf averages
`chit`	thermostat momentum
`chip`	barostat momentum
`conint`	conserved quantity for selected ensemble
`nzden`	number of configurations used in z density

record 2:

`virtot`	total system virial
`vircom`	rigid body COM virial
`eta`	scaling factors for simulation cell matrix elements (9)
`strcns`	constraint stress tensor elements (9)
`strbod`	rigid body stress tensor elements (9)

record 3:

`stpval`	instantaneous values of thermodynamic variables (`mxnstk`)

record 4:

`sumval`	average values of thermodynamic variables (`mxnstk`)

record 5:

`ssqval`	fluctuation (squared) of thermodynamic variables (`mxnstk`)

record 6:

`zumval`	running totals of thermodynamic variables (`mxnstk`)

record 7:

`ravval`	rolling averages of thermodynamic variables (`mxnstk`)

record 8:

`stkval`	stacked values of thermodynamic variables (`mxstak`×`mxnstk`)

record 9:

`xx0`	x component of atomic displacement (MSD) (`mxatms`)

yy0	y component of atomic displacement (MSD) (mxatms)
zz0	z component of atomic displacement (MSD) (mxatms)
record 10:	
xxs	x-coordinates of tether points (mxatms)
yys	y-coordinates of tether points (mxatms)
zzs	z-coordinates of tether points (mxatms)
record 11:	
rdf	(Optional) RDF array (mxrdf×mxvdw)
record 12:	
zdens	(Optional) z-density array (mxrdf×mxsvdw)

4.1.4.2 Further Comments

Note that recompiling DL_POLY_2 with a different DL_PARAMS.INC file, may render any existing REVOLD file unreadable by the code.

4.1.5 The TABLE File

The TABLE file provides an alternative way of reading in the short range potentials - in tabular form. This is particularly useful if an analytical form of the potential does not exist or is too complicated to specify in the FORGEN subroutine. The table file is read by the subroutine FORTAB.F in the VDW_TERMS.F file..

The option of using tabulated potentials is specified in the FIELD file (see above). The specific potentials that are to be tabulated are indicated by the use of the **tab** keyword on the record defining the short range potential (see table 4.12). The directive **vdwtable** may be used in place of **vdw** to indicate that one or more of the short ranged potentials is specified in the form of a table.

4.1.5.1 Format

The file is fixed-formatted with integers as "i10", reals as "e15.8". Character variables are read as "a8". The header record is formatted as 80 alphanumeric characters.

4.1.5.2 Definitions of Variables

record 1		
header	a80	file header
record 2		
delpot	real	mesh resolution in Å
cutpot	real	cutoff used to define tables Å
ngrid	integer	number of grid points in tables

The subsequent records define each tabulated potential in turn, in the order indicated by the specification in the FIELD file. Each potential is defined by a header record and a set of data records with the potential and force tables.

header record:

`atom 1`	a8	first atom type
`atom 2`	a8	second atom type

potential data records: (*number of data records* = Int((`ngrid`+3)/4))

`data 1`	real	data item 1
`data 2`	real	data item 2
`data 3`	real	data item 3
`data 4`	real	data item 4

force data records: (*number of data records* = Int((`ngrid`+3)/4))

`data 1`	real	data item 1
`data 2`	real	data item 2
`data 3`	real	data item 3
`data 4`	real	data item 4

4.1.5.3 Further Comments

It should be noted that the number of grid points in the TABLE file should not be less than the number of grid points DL_POLY_2 is expecting. (This number is given by the parameter `mxgrid`, which is defined in the PARSET.F subroutine in the SETUP_PROGRAM.F file.) DL_POLY_2 will re-interpolate the tables if `ngrid`≥`mxgrid`, but will abort if `ngrid`<`mxgrid`.

The potential and force tables are used to fill the internal arrays `vvv` and `ggg` respectively (see section 2.3.1). The contents of force arrays are derived from the potential via the formula:

$$G(r) = -r\frac{\partial}{\partial r}U(r).$$

Note this is *not* the same as the true force.

Important The potential and force arrays in the TABLE file are written in the same units as the FIELD file. So if you specified a particular unit using the **UNITS** directive in the FIELD file, the same units are expected here. It is useful to note that the definition of the force arrays given above means that the units are the same as for the potential - i.e. are handled using the same conversion factors.

4.1.6 The TABEAM File

The TABEAM file contains the tabulated potential functions (no explicit analytic form) describing the EAM metal interactions in the MD system. This file is read by the subroutine METTAB.

The EAM potential for an n component metal alloy requires the specification of n electron density functions (one for each atom type) and n embedding functions (again one for each atom type) and $n(n+1)/2$ *cross* pair potential functions. This makes $n(n+5)/2$

functions in total. **Note** that the option of using EAM interactions must also be *explicitly* declared in the FIELD file so that for the n component alloy there are $n(n+1)/2$ *cross* pair potential (**eam**) keyword entries in FILED (see above). (**Note** that all metal interactions must be of the same type!)

4.1.6.1 The TABEAM File Format

The file is free-formatted but blank and commented lines are not allowed.

4.1.6.2 Definitions of Variables

record 1

`header`	a100	file header

record 2

`numpot`	integer	number of potential functions in file

The subsequent records define the $n(n+5)/2$ functions for an n component alloy - n electron density functions (one for each atom type) - **dens**ity keyword, n embedding functions (again one for each atom type) - **embe**ding keyword, and $n(n+1)/2$ *cross* pair potential functions - **pair**s keyword. The functions may appear in any random order in TABEAM as their identification is based on their unique keyword, defined first in the function's header record. The header record is followed by a predefined number of data records **as a maximum of four data per record are read in** - allowing for incompletion of the very last record.

header record:

keyword	a4	type of EAM function: **dens**, **embe**d or **pair**
`atom 1`	a8	first atom type
`atom 2`	a8	second atom type - only specified for **pair** potential functions
`ngrid`	integer	number of function data points to read in
`limit 1`	real	lower interpolation limit in Åfor **dens** and **pair** or in density units for **embe**d
`limit 2`	real	upper interpolation limit in Åfor **dens** and **pair** or in density units for **embe**d

function data records: (*number of data records* = `Int`((`ngrid`+3)/4))

`data 1`	real	data item 1
`data 2`	real	data item 2
`data 3`	real	data item 3
`data 4`	real	data item 4

4.2 The OUTPUT Files

DL_POLY_2 produces up to seven output files: HISTORY, OUTPUT, REVCON, REVIVE, RDFDAT, ZDNDAT and STATIS. These respectively contain: a dump file of atomic coordinates, velocities and forces; a summary of the simulation; the restart configuration; statistics accumulators; radial distribution data, Z-density data and a statistical history.

4.2.1 The HISTORY File

The HISTORY file is the dump file of atomic coordinates, velocities and forces. Its principal use is for off-line analysis. The file is written by the subroutines TRAJECT or TRAJECT_U. The control variables for this file are `ltraj, nstraj, istraj` and `keytrj` which are created internally, based on information read from the **traj** directive in the CONTROL file (see above). The HISTORY file will be created only if the directive **traj** appears in the CONTROL file. Note that the HISTORY file can be written in either a formatted or unformatted version. We describe each of these separately below.

The HISTORY file can become *very* large, especially if it is formatted. For serious simulation work it is recommended that the file be written to a scratch disk capable of accommodating a large data file. Alternatively the file may be written as unformatted (below), which has the additional advantage of speed. However, writing an unformatted file has the disadvantage that the file may not be readily readable except by the machine on which it was created. This is particularly important if graphical processing of the data is required.

4.2.1.1 The Formatted HISTORY File

The formatted HISTORY file is written by the subroutine TRAJECT and has the following structure.

record 1 (a80)

`header`	a80	file header

record 2 (3i10)

`keytrj`	integer	trajectory key (see table 4.3)
`imcon`	integer	periodic boundary key (see table 4.6)
`natms`	integer	number of atoms in simulation cell

For timesteps greater than `nstraj` the HISTORY file is appended at intervals specified by the **traj** directive in the CONTROL file, with the following information for each configuration:

record i (a8,4i10,f12.6)

`timestep`	a8	the character string “timestep”
`nstep`	integer	the current time-step
`natms`	integer	number of atoms in configuration
`keytrj`	integer	trajectory key (again)

`imcon`	integer	periodic boundary key (again)
`tstep`	real	integration timestep
record ii (3g12.4) for `imcon` > 0		
`cell(1)`	real	x component of a cell vector
`cell(2)`	real	y component of a cell vector
`cell(3)`	real	z component of a cell vector
record iii (3g12.4) for `imcon` > 0		
`cell(4)`	real	x component of b cell vector
`cell(5)`	real	y component of b cell vector
`cell(6)`	real	z component of b cell vector
record iv (3g12.4) for `imcon` > 0		
`cell(7)`	real	x component of c cell vector
`cell(8)`	real	y component of c cell vector
`cell(9)`	real	z component of c cell vector

This is followed by the configuration for the current timestep. i.e. for each atom in the system the following data are included:

record a (a8,i10,2f12.6)		
`atmnam`	a8	atomic label
`iatm`	i10	atom index
`weight`	f12.6	atomic mass (a.m.u.)
`charge`	f12.6	atomic charge (e)
record b (3e12.4)		
`xxx`	real	x coordinate
`yyy`	real	y coordinate
`zzz`	real	z coordinate
record c (3e12.4) only for `keytrj` > 0		
`vxx`	real	x component of velocity
`vyy`	real	y component of velocity
`vzz`	real	z component of velocity
record d (3e12.4) only for `keytrj` > 1		
`fxx`	real	x component of force
`fyy`	real	y component of force
`fzz`	real	z component of force

Thus the data for each atom is a minimum of two records and a maximum of 4.

4.2.1.2 The Unformatted HISTORY File

The unformatted HISTORY file is written by the subroutine TRAJECT_U and has the following structure:

record 1
`header` configuration name (character*80)
record 2
`natms` number of atoms in the configuration (real*8)
record 3
`atname(1,...,natms)` atom names or symbols (character*8)
record 4
`weight(1,...,natms)` atomic masses (real*8)
record 5
`charge(1,...,natms)` atomic charges (real*8)

For time-steps greater than `nstraj`, the HISTORY file is appended, at intervals specified by the **traj** directive in the CONTROL file, with the following information:

record i
`nstep` the current time-step (real*8)
`natms` number of atoms in configuration (real*8)
`keytrj` trajectory key (real*8)
`imcon` image convention key (real*8)
`tstep` integration timestep (real*8)
record ii for `imcon` > 0
`cell(1,...,9)` a, b and c cell vectors (real*8)
record iii
`xxx(1,...,natms)` atomic x-coordinates (real*8)
record iv
`yyy(1,...,natms)` atomic y-coordinates (real*8)
record v
`zzz(1,...,natms)` atomic z-coordinates (real*8)
record vi only for `keytrj`> 0
`vxx(1,...,natms)` atomic velocities x-component (real*8)
record vii only for `keytrj`> 0
`vyy(1,...,natms)` atomic velocities y-component (real*8)
record viii only for `keytrj`> 0
`vzz(1,...,natms)` atomic velocities z-component (real*8)
record ix only for `keytrj`> 1
`fxx(1,...,natms)` atomic forces x-component (real*8)
record x only for `keytrj`> 1
`fyy(1,...,natms)` atomic forces y-component (real*8)
record xi only for `keytrj`> 1
`fzz(1,...,natms)` atomic forces z-component (real*8)

Note the implied conversion of integer variables to real on record i.

4.2.2 The OUTPUT File

The job output consists of 7 sections: Header; Simulation control specifications; Force field specification; Summary of the initial configuration; Simulation progress; Summary of statistical data; Sample of the final configuration; and Radial distribution functions. These sections are written by different subroutines at various stages of a job. Creation of the OUTPUT file *always* results from running DL_POLY_2 . It is meant to be a human readable file, destined for hardcopy output.

4.2.2.1 Header

Gives the DL_POLY_2 version number, the number of processors used and a title for the job as given in the header line of the input file CONTROL. This part of the file is written from the subroutines DLPOLY and SIMDEF

4.2.2.2 Simulation Control Specifications

Echoes the input from the CONTROL file. Some variables may be reset if illegal values were specified in the CONTROL file. This part of the file is written from the subroutine SIMDEF.

4.2.2.3 Force Field Specification

Echoes the FIELD file. A warning line will be printed if the system is not electrically neutral. This warning will appear immediately before the non-bonded short-range potential specifications. This part of the file is written from the subroutine SYSDEF.

4.2.2.4 Summary of the Initial Configuration

This part of the file is written from the subroutine SYSGEN. It states the periodic boundary specification, the cell vectors and volume (if appropriate) and the initial configuration of (a maximum of) 20 atoms in the system. The configuration information given is based on the value of `levcfg` in the CONFIG file. If `levcfg` is 0 (or 1) positions (and velocities) of the 20 atoms are listed. If `levcfg` is 2 forces are also written out.

For periodic systems this is followed by the long range corrections to the energy and pressure.

4.2.2.5 Simulation Progress

This part of the file is written by the DL_POLY_2 root segment DLPOLY. The header line is printed at the top of each page as:

```
------------------------------------------------------------------------------------------------

    step   eng_tot  temp_tot   eng_cfg   eng_vdw   eng_cou   eng_bnd   eng_ang   eng_dih   eng_tet
time(ps)    eng_pv  temp_rot   vir_cfg   vir_vdw   vir_cou   vir_bnd   vir_ang   vir_con   vir_tet
cpu  (s)    volume  temp_shl   eng_shl   vir_shl     alpha      beta   gamma   vir_pmf     press
```

The labels refer to :

line 1

step	MD step number
eng_tot	total internal energy of the system
temp_tot	system temperature
eng_cfg	configurational energy of the system
eng_vdw	configurational energy due to short-range potentials
eng_cou	configurational energy due to electrostatic potential
eng_bnd	configurational energy due to chemical bond potentials
eng_ang	configurational energy due to valence angle and three-body potentials
eng_dih	configurational energy due to dihedral inversion and four-body potentials
eng_tet	configurational energy due to tethering potentials

line 2

time(ps)	elapsed simulation time (ps) since the beginning of the job
eng_pv	enthalpy of system
temp_rot	rotational temperature
vir_cfg	total configurational contribution to the virial
vir_vdw	short range potential contribution to the virial
vir_cou	electrostatic potential contribution to the virial
vir_bnd	chemical bond contribution to the virial
vir_ang	angular and three body potentials contribution to the virial
vir_con	constraint bond contribution to the virial
vir_tet	tethering potential contribution to the virial

line 3

cpu (s)	elapsed cpu time since the beginning of the job
volume	system volume
temp_shl	core-shell temperature
eng_shl	configurational energy due to core-shell potentials
vir_shl	core-shell potential contribution to the virial
alpha	angle between b and c cell vectors
beta	angle between c and a cell vectors
gamma	angle between a and b cell vectors
vir_pmf	Potential of mean force constraint contribution to the virial
press	pressure

Note: The total internal energy of the system (variable `tot_energy`) includes all contributions to the energy (including system extensions due to thermostats etc.) It is nominally the *conserved variable* of the system, and is not to be confused with conventional system energy, which is a sum of the kinetic and configuration energies.

The interval for printing out these data is determined by the directive **print** in the CONTROL file. At each time-step that printout is requested the instantaneous values of

the above statistical variables are given in the appropriate columns. Immediately below these three lines of output the rolling averages of the same variables are also given. The maximum number of time-steps used to calculate the rolling averages is determined by the parameter `mxstak` defined in the PARSET.F subroutine of the SETUP_PROGRAM./F file. The working number of time-steps for rolling averages is controlled by the directive **stack** in file CONTROL (see above). The default value is `mxstak`.

Energy Units: The energy unit for the data appearing in the OUTPUT is defined by the **units** directive appearing in the CONTROL file.

Pressure units: The unit of pressure is k *atm*, irrespective of what energy unit is chosen.

4.2.2.6 Summary of Statistical Data

This portion of the OUTPUT file is written from the subroutine RESULT. The number of time-steps used in the collection of statistics is given. Then the averages over the production portion of the run are given for the variables described in the previous section. The root mean square variation in these variables follow on the next two lines. The energy and pressure units are as for the preceeding section.

Also provided in this section is an estimate of the diffusion coefficient for the different species in the simulation, which is determined from a *single time origin* and is therefore very approximate. Accurate determinations of the diffusion coefficients can be obtained using the MSD utility program, which processes the HISTORY file (see chapter 6).

If an NPT or N$\underline{\underline{\sigma}}$T simulation is performed the OUTPUT file also provides the mean stress (pressure) tensor and mean simulation cell vectors.

4.2.2.7 Sample of Final Configuration

The positions, velocities and forces of the 20 atoms used for the sample of the initial configuration (see above) are given. This is written by the subroutine RESULT.

4.2.2.8 Radial Distribution Functions

If both calculation and printing of radial distribution functions have been requested (by selecting directives **rdf** and **print rdf** in the CONTROL file) radial distribution functions are printed out. This is written from the subroutine RDF1. First the number of time-steps used for the collection of the histograms is stated. Then each function is given in turn. For each function a header line states the atom types ('a' and 'b') represented by the function. Then r, $g(r)$ and $n(r)$ are given in tabular form. Output is given from 2 entries before the first non-zero entry in the $g(r)$ histogram. $n(r)$ is the average number of atoms of type 'b' within a sphere of radius r around an atom of type 'a'. Note that a readable version of these data is provided by the RDFDAT file (below).

4.2.2.9 Z Density Profile

If both calculation and printing of Z density profiles has been requested (by selecting directives **zden** and **print rdf** in the CONTROL file Z density profiles are printed out as the last part of the OUTPUT file. This is written by the subroutine ZDEN1. First the number of time-steps used for the collection of the histograms is stated. Then each function is given in turn. For each function a header line states the atom type represented by the function. Then z, $\rho(z)$ and $n(z)$ are given in tabular form. Output is given from $Z = [-L/2, L/2]$ where L is the length of the MD cell in the Z direction and $\rho(z)$ is the mean number density. $n(z)$ is the running integral from $-L/2$ to z of (xy cell area)$\rho(s)ds$. Note that a readable version of these data is provided by the ZDNDAT file (below).

4.2.3 The REVCON File

This file is formatted and written by the subroutine REVIVE. REVCON is the restart configuration file. The file is written every `ndump` time steps in case of a system crash during execution and at the termination of the job. A successful run of DL_POLY_2 will always produce a REVCON file, but a failed job may not produce the file if an insufficient number of timesteps have elapsed. `ndump` is a parameter defined in the PARSET.F subroutine of the SETUP_PROGRAM.F file found in the *srcf90* directory of DL_POLY_2 . Changing `ndump` necessitates recompiling DL_POLY_2 . REVCON is identical in format to the CONFIG input file (see section 4.1.2). REVCON should be renamed CONFIG to continue a simulation from one job to the next. This is done for you by the *copy* macro supplied in the *execute* directory of DL_POLY_2 .

4.2.4 The REVIVE File

This file is unformatted and written by the subroutine REVIVE. It contains the accumulated statistical data. It is updated whenever the file REVCON is updated (see previous section). REVIVE should be renamed REVOLD to continue a simulation from one job to the next. This is done by the *copy* macro supplied in the *execute* directory of DL_POLY_2 . In addition, to continue a simulation from a previous job the **restart** keyword must be included in the CONTROL file.

The format of the REVIVE file is identical to the REVOLD file described in section 4.1.4.

4.2.5 The RDFDAT File

This is a formatted file containing em Radial Distribution Function (RDF) data. Its contents are as follows:

record 1

`cfgname`	character (A80)	configuration name

record 2

`ntpvdw`	integer (i10)	number of RDFs in file

`mxrdf`	integer (i10)	number of data points in each RDF

There follow the data for each individual RDF i.e. *ntpvdw* times. The data supplied are as follows:

first record

`atname 1`	character (A8)	first atom name
`atname 2`	character (A8)	second atom name

following records (*mxrdf* records)

`radius`	real (e14)	interatomic distance (A)
`g(r)`	real (e14)	RDF at given radius.

Note the RDFDAT file is optional and appears when the **print rdf** option is specified in the CONTROL file.

4.2.6 The ZDNDAT File

This is a formatted file containing the Z-density data. Its contents are as follows:

record 1

`cfgname`	character (A80)	configuration name

record 2

`mxrdf`	integer (i10)	number of data points in the Z-density function

following records (*mxrdf* records)

`z`	real (e14)	distance in z direction (A)
$\rho(z)$	real (e14)	Z-density at given height `z`

Note the ZDNDAT file is optional and appears when the **print rdf** option is specified in the CONTROL file.

4.2.7 The STATIS File

The file is formatted, with integers as "i10" and reals as "e14.6". It is written by the subroutine STATIC. It consists of two header records followed by many data records of statistical data.

record 1

`cfgname`	character	configuration name

record 2

`string`	character	energy units

Data records

Subsequent lines contain the instantaneous values of statistical variables dumped from the array `stpval`. A specified number of entries of `stpval` are written in the format "(1p,5e14.6)". The number of array elements required (determined by the parameter `mxnstk` in the DL_PARAMS.INC file) is

$$\begin{aligned} \texttt{mxnstk} \geq \quad & 27 + \texttt{ntpatm}(\text{number of unique atomic sites}) \\ & +9(\text{if stress tensor calculated}) \\ & +9(\text{if constant pressure simulation requested}) \end{aligned}$$

The STATIS file is appended at intervals determined by the **stats** directive in the CONTROL file. The energy unit is as specified in the CONTROL file with the the **units** directive, and are compatible with the data appearing in the OUTPUT file. The contents of the appended information is:

record i		
`nstep`	integer	current MD time-step
`time`	real	elapsed simulation time = `nstep`$\times\Delta t$
`nument`	integer	number of array elements to follow
record ii `stpval`(1) – `stpval`(5)		
`engcns`	real	total extended system energy (i.e. the conserved quantity)
`temp`	real	system temperature
`engcfg`	real	configurational energy
`engsrp`	real	VdW/metal/Tersoff energy
`engcpe`	real	electrostatic energy
record iii `stpval`(6) – `stpval`(10)		
`engbnd`	real	chemical bond energy
`engang`	real	valence angle/3-body potential energy
`engdih`	real	dihedral/inversion/four body energy
`engtet`	real	tethering energy
`enthal`	real	enthalpy (total energy + PV)
record iv `stpval`(11) – `stpval`(15)		
`tmprot`	real	rotational temperature
`vir`	real	total virial
`virsrp`	real	VdW/metal/Tersoff virial
`vircpe`	real	electrostatic virial
`virbnd`	real	bond virial
record v `stpval`(16) -`stpval`(20)		
`virang`	real	valence angle/3-body virial
`vircon`	real	constraint virial
`virtet`	real	tethering virial
`volume`	real	volume
`tmpshl`	real	core-shell temperature

record vi `stpval`(21) `-stpval`(25)		
`engshl`	real	core-shell potential energy
`virshl`	real	core-shell virial
`alpha`	real	MD cell angle α
`beta`	real	MD cell angle β
`gamma`	real	MD cell angle γ
record vii `stpval`(26) `-stpval`(27)		
`virpmf`	real	Potential of Mean Force virial
`press`	real	pressure
the next `ntpatm` **entries**		
`amsd(1)`	real	mean squared displacement of first atom types
`amsd(2)`	real	mean squared displacement of second atom types
`...`	...	...
`amsd(ntpatm)`	real	mean squared displacement of last atom types
the next 9 entries - ***if*** **the stress tensor is calculated**		
`stress(1)`	real	xx component of stress tensor
`stress(2)`	real	xy component of stress tensor
`stress(3)`	real	xz component of stress tensor
`stress(4)`	real	yx component of stress tensor
`...`	real	...
`stress(9)`	real	zz component of stress tensor
the next 9 entries - ***if*** **a NPT simulation is undertaken**		
`cell(1)`	real	x component of a cell vector
`cell(2)`	real	y component of a cell vector
`cell(3)`	real	z component of a cell vector
`cell(4)`	real	x component of b cell vector
`...`	real	...
`cell(9)`	real	z component of c cell vector

Note. The stress tensor is calculated only if the code is compiled with the "-STRESS" option (see section 3.2.1). Cell shape varying constant pressure simulations (keyword **ensemble nst ...** in the CONTROL file) are only possible if the stress tensor is calculated. If isotropic constant pressure simulations are required, where the cell volume but not the shape may vary, (keyword **ensemble npt ...**) the stress tensor need not be calculated.

Chapter 5

DL_POLY_2 Examples

Scope of Chapter

This chapter describes the standard test cases for DL_POLY_2 , the input and output files for which are in the *data* sub-directory.

5.1 DL_POLY Examples

5.1.1 Test Cases

The following example data sets (both input and output) are stored in the subdirectory *data*. Two versions are provided for the Leapfrog (LF) and Velocity Verlet (VV) algorithms respectively, so that you may check that your version of DL_POLY is working correctly. All the jobs are short and should require no more than a few minutes execution time, even on a single processor computer. he test cases can be chosen by typing

select n a

from the *execute* directory, where n is the number of the test case and a is either LF or VV. The *select* macro will copy the appropriate CONTROL, CONFIG, and FIELD files to the *execute* directory ready for execution. The output files OUTPUT, REVCON and STATIS may be compared with the files supplied in the *data* directory.

The example output files provided in the *data* directory were obtained on 8 processors of a Cray XD1 parallel system. The program was compiled with the three point interpolation option.

It should be noted that the potentials and the simulation conditions used in the following test cases are chosen to demonstrate functionality only. **They are not necessarily appropriate for serious simulation of the test systems.** Note also that the DL_POLY_2 Graphical User Interface [8] provides a convenient means for running and viewing these test cases.

5.1.1.1 Test Case 1: $KNaSi_2O_5$

Potassium Sodium disilicate glass ($NaKSi_2O_5$) using two and three body potentials. Some of the two body potentials are read from the TABLE file. Electrostatics are handled by a multiple timestep Ewald sum method. Cubic periodic boundaries are in use. NVE ensemble.

5.1.1.2 Test Case 2: Metal simulation with Sutton Chen potentials

FCC Aluminium using Sutton-Chen potentials. Temperature is controlled by the method of Gaussian constraints. NVT Evans ensemble.

5.1.1.3 Test Case 3: An antibiotic in water

Valinomycin in 1223 spc water molecules. The temperature is controlled by a Nosé-Hoover thermostat while electrostatics are handled by a shifted Coulombic potential. The water is defined as a rigid body while bond constraints are applied to all chemical bonds in the valinomycin. Truncated octahedral boundary conditions are used. NVT Hoover ensemble.

5.1.1.4 Test Case 4: Shell model of water

256 molecules of water with a polarizable oxygen atom using adiabatic dynamics. Temperature is controlled by the Berendsen thermostat while electrostatics are handled by the

reaction field method with a "charge group" cutoff scheme. "Slab" period boundary conditions are used. The water molecule (apart from the shell) is treated as a rigid body. NVT Berendsen 'ensemble'.

5.1.1.5 Test Case 5: Shell model of $MgCl_2$ at constant pressure

Adiabatic shell model simulation of $MgCl_2$. Temperature and pressure are controlled by a Berendsen thermostat and barostat. An Ewald sum is used with cubic periodic boundary conditions. NPT Berendsen 'ensemble'.

5.1.1.6 Test Case 6: PMF calculation

Potential of mean force calculation of a potassium ion in SPC water. Electrostatics are handled by the Ewald sum. The water is treated as a constrained triangle. PMF 'ensemble'

5.1.1.7 Test Case 7: Linked rigid bodies

8 biphenyl molecules in cubic boundary conditions. Each phenyl ring is treated as a rigid body, with a constraint bond to the other ring of the molecule. In the centre of each ring are three massless charge sites which imparts a quadrupole moment to the ring. NVE ensemble.

5.1.1.8 Test Case 8: An osmosis experiment with a semi permeable membrane

The membrane is a collection of tethered sites interconnected by harmonic springs. There are no electrostatic forces in the system. The simulation is run with the Hoover anisotropic constant presure algorithm. (NST Hoover ensemble.)

5.1.1.9 Test Case 9: A surfactant at the air-water interface

The system is comprised of 32 surfactant molecules (trimethylaminododecane bromide or TAB-C12) arranged either side of a slab of 342 water molecules approximately 30 $\AA$ thick. The surfactant chains are treated with rigid bonds and the water molecules are treated as rigid bodies. The TAB headgroup has fractional charges summing to +1 (the bromide ion has charge -1). The Ewald sum handles the electrostatic calculations. The short range forces are taken from the Dreiding force field. NVE ensemble.

5.1.1.10 Test Case 10: DNA strand in water

This system consists of a strand of DNA 1260 atoms in length in a solution of 706 (SPC) water molecules. The DNA is aligned in the Z-direction and *hexagonal prism* periodic boundary conditions applied. The electrostatic interactions are calculated using the Smoothed Particle Mesh Ewald method. Note that the system has a strong overall negative charge which is strongly anisotropic in distribution. The short range forces are taken from the Dreiding force field, and constraints are used for all covalent bonds. For simplicity H-bonds are treated as harmonic bonds with an equilibrium bondlength of 1.724 $\AA$. NVE ensemble.

5.1.1.11 Test Case 11: Hautman-Klein test case 1

The system consists of 100 short chain surfactant molecules in a layer simulated under NVE conditions . The total system size is 2300 atoms and the XY periodicity is a square. The Dreiding force field describes the molecular interactions. All bonds are harmonic and all atoms are explicit. The link-cell algorithm is in operation. NVE ensemble.

5.1.1.12 Test Case 12: Hautman-Klein test case 2

This is a simple test system consisting of 1024 charged particles in a layer under NVE conditions. Lennard Jones forces are used to keep the atoms apart. The similation cell is square in the XY plane. NVE ensemble.

5.1.1.13 Test Case 13: Carbon Nanotube with Tersoff potential

This system consists of 800 carbon atoms in a nanotube 41.7 A in length. The MD cell is orthorhombic and square in the XY plane. The integration algorithm is NPT Berendsen. This is a test for the Tersoff potential . NPT Berendsen 'ensemble'.

5.1.1.14 Test Case 14: Carbon Diamond with Tersoff potential

This is another test of the Tersoff potential , this time for the carbon diamond structure consisting of 512 atoms. A cubic MD cell is used with a NST Hoover integration algorithm. NST Hoover ensemble.

5.1.1.15 Test Case 15: Silicon Carbide with Tersoff potential

This is an alloy system consisting of 2744 atoms of silicon carbide in a diamond structure. The potential function used is the Tersoff potential . The integration algorithm is NPT Hoover and the initial MD cell is cubic. NPT Hoover ensemble.

5.1.1.16 Test Case 16: Magnesium Oxide with relaxed shell model

Relaxed shell model of magnesium oxide with 324 sites. The lattice is cubic and the integration algorithm is NST Berendsen. NST Berendsen 'ensemble'.

5.1.1.17 Test Case 17: Sodium ion in SPC water

A simple simulation of a sodium ion in 140 SPC water molecules (421 sites in all). The water molecules are treated as rigid bodies.The algorithm is the NVE ensemble and the Ewald sum handles the electrostatic forces. The MD box is cubic. NVE ensemble.

5.1.1.18 Test Case 18: Sodium chloride molecule in SPC water

This system resembles test case 17, except that a sodium chloride ion pair is dissolved in 139 SPC water molecules (419 sites in all). The MD cell is cubic and the water molecules

are treated by constraint dynamics in the NVE Evans scheme. Ewald's method handles the electrostatics. NVT Evans ensemble.

5.1.1.19 Test Case 19: Sodium chloride molecule in SPC water

This is a repeat of test case 18, except that half of the water molecules are treated using constraint dynamics and the rest by rigid body dynamics. The integration algorithm is NPT Hoover. NPT Hoover ensemble.

5.1.1.20 Test Case 20: Linked benzene ring molecules

This test consists of pairs of benzene rings linked via a rigid (constraint) bond. Each molecule has 22 atoms and there are 81 molecules, making a total of 1782 sites. The benzene rings are treated in a variety of ways in the same system. In one third of cases the benzene rings and hydrogens form rigid groups. In another third the carbon rings are rigid but the C-H bonds are treated via constraints. In the final third, the C-H bonds are fully flexible and the rings are rigid. The MD cell is orthorhombic (nearly cubic) and the integration is NPT hoover. NPT Hoover ensemble.

5.1.1.21 Test Case 21: Aluminium metal with EAM potential

This case presents an example of the use of the EAM potential for metals, in this case aluminium. The system is 256 atoms and runs under a berendsen NPT enemble.

5.1.1.22 Test Case 22: Copper metal with EAM potential

Another example of a metal with an EAM potential. 256 copper atoms under a Berendsen NPT ensemble.

5.1.1.23 Test Case 23: Copper-Gold (3/1) alloy with Gupta potential

This is an example of the analytical Gupta potential applied to a copper-gold alloy with a 3/1 Cu/Au ratio. The system consists of 256 atoms in total running under the NVE ensemble.

5.1.1.24 Test Case 24: Iron metal with Finnis Sinclair potential

In this example the analytical Finnis-Sinclair potential is applied to iron. The system consists of 250 iron atoms and runs under a Berendsen NPT ensemble.

5.1.1.25 Test Case 25: Nickel-Aluminium (1/1) alloy with EAM potential

Another example of an alloy using the EAM potential. This is a Nickel-Aluminium alloy in the 1/1 ratio. The NVE ensemble is used and the system has 432 atoms.

5.1.1.26 Test Case 26: Nickel metal with EAM potential

Another EAM simulation of a metal. 256 Nickel atoms under the Berendsen NPT ensemble.

5.1.2 Benchmark Cases

These represent rather larger test cases for DL_POLY_2 that are also suitable for benchmarking the code on large scale computers. They have been selected to show fairly the the capabilities and limitations of the code.

5.1.2.1 Benchmark 1

Simulation of metallic aluminium at 300K using a Sutton-Chen density dependent potential. The system is comprised of 19652 identical atoms. The simulation runs on 16 to 512 processors only.

5.1.2.2 Benchmark 2

Simulation of a 15-peptide in 1247 water molecules. This was designed as an AMBER comparison. The system consists of 3993 atoms in all and runs on 8-512 processors. It uses neutral group electrostatics and rigid bond constraints and is one of the smallest benchmarks in the set.

5.1.2.3 Benchmark 3

Simulation of the enzyme transferrin in 8102 water molecules. The simulation makes use of neutral group electrostatics and rigid bond constraints. The system is 27539 atoms and runs on 8-512 processors.

5.1.2.4 Benchmark 4

Simulation of a sodium chloride melt with Ewald sum electrostatics and a multiple timestep algorithm to enhance performance. The system is comprised of 27000 atoms and runs on 8-512 processors.

5.1.2.5 Benchmark 5

Simulation of a sodium-potassium disilicate glass. Uses Ewald sum electrostatics, a multiple timestep algorithm and a three-body valence angle potentials to support the silicate structure. It also using tabluated two-body potentials stored in the file TABLE. The system is comprised of 8640 atoms and runs on 16-512 processors.

5.1.2.6 Benchmark 6

Simulation of a potassium-valinomycin complex in 1223 water molecules using an adapted AMBER forcefield and truncated octahedral periodic boundary conditions. The system size is 3838 atoms and runs on 16-512 processors.

5.1.2.7 Benchmark 7

Simulation of gramicidin A molecule in 4012 water molecules using neutral group electrostatics. The system is comprised of 12390 atoms and runs on 8-512 processors. This example was provided by Lewis Whitehead at the University of Southampton.

5.1.2.8 Benchmark 8

Simulation of an isolated magnesium oxide microcrystal comprised of 5416 atoms originally in the shape of a truncated octahedron. Uses full coulombic potential. Runs on 16-512 processors.

5.1.2.9 Benchmark 9

Simulation of a model membrane with 196 41-unit membrane chains, 8 valinomycin molecules and 3144 water molecules using an adapted AMBER potential, multiple timestep algorithm and Ewald sum electrostatics. The system is comprised of 18866 atoms and runs on 8-512 processors.

Chapter 6

DL_POLY_2 Utilities

Scope of Chapter

This chapter describes the more important utility programs and subroutines of DL_POLY_2 , found in the sub-directory *utility*. A more complete description of the sub-directory contents is to be found in the DL_POLY_2 Reference Manual.

6.1 Miscellaneous Utilities

6.1.1 Useful Macros

6.1.1.1 Macros

Macros are simple executable files containing standard unix commands. A number of the are supplied with DL_POLY and are found in the *execute* sub-directory. The available macros are as follows.

- *cleanup*
- *copy*
- *gopoly*
- *gui*
- *select*
- *store*

The function of each of these is described below. It is worth noting that most of these functions can be performed by the DL_POLY_2 java GUI [8].

6.1.1.2 *cleanup*

cleanup removes several standard data files from the *execute* sub-directory. It contains the unix commands:

```
rm OUTPUT REVCON REVOLD STATIS REVIVE gopoly.*
```

and removes the files OUTPUT, REVCON, REVOLD, STATIS, REVIVE and gopoly.* (all variants). It is useful for cleaning the sub-directory up after a run. (Useful data should be stored elsewhere however!)

6.1.1.3 *copy*

copy invokes the unix commands:

```
mv CONFIG CONFIG.OLD
mv REVCON CONFIG
mv REVIVE REVOLD
```

which collectively prepare the DL_POLY files in the *execute* sub-directory for the continuation of a simulation. It is always a good idea to store these files elsewhere in addition to using this macro.

6.1.1.4 *gopoly*

gopoly is used to submit a DL_POLY job to the Daresbury IBM SP/2, which operates a LOADLEVELLER job queuing system. It invokes the following script.

```
#@ min_processors = 4
#@ max_processors = 4
#@ job_type = parallel
#@ requirements = (Adapter == "hps_ip") && ( Pool == 2)
#@ executable = /usr/bin/poe
#@ cpu_limit = 00:10:00
#@ arguments = ~/dl_poly_2.10/execute/DLPOLY.X -euilib ip
#@ output = gopoly.o
#@ error = gopoly.e
#@ class = dev
#@ queue
```

Using LOADLEVELLER, the job is submitted by the unix command:

llsubmit gopoly

where *llsubmit* is a local command for submission to the SP/2. The number of required nodes and the job time are indicated in the above script.

6.1.1.5 *gui*

gui is a macro that starts up the DL_POLY_2 Java GUI. It invokes the following unix commands:

```
java -jar ../java/GUI.jar
```

In other words the macro invokes the Java Virtual Machine which executes the instructions in the Java archive file GUI.jar, which is stored in the *java* subdirectory of DL_POLY_2 . (Note: Java 1.3.0 or a higher version is required to run the GUI.)

6.1.1.6 *select*

select is a macro enabling easy selection of one of the test cases. It invokes the unix commands:

```
cp ../data/TEST$1/$2/CONTROL CONTROL
cp ../data/TEST$1/$2/FIELD   FIELD
cp ../data/TEST$1/$2/CONFIG  CONFIG
cp ../data/TEST$1/$2/TABLE   TABLE
```

select requires two arguments to be specified:

select n a

where *n* is the (integer) test case number, which ranges from 1 to 20 and *a* is the character string LF or VV according to which algorithm leapfrog (LF)or velcioty Verlet (VV) is required.

This macro sets up the required input files in the *execute* sub-directory to run the *n*-th test case.

6.1.1.7 *store*

The *store* macro provides a convenient way of moving data back from the *execute* sub-directory to the *data* sub-directory. It invokes the unix commands:

```
mkdir ../data/TEST$1
mkdir ../data/TEST$1/$2
mv OUTPUT ../data/TEST$1/$2/OUTPUT
mv REVCON ../data/TEST$1/$2/REVCON
mv STATIS ../data/TEST$1/$2/STATIS
mv REVIVE ../data/TEST$1/$2/REVIVE
mv RDFDAT ../data/TEST$1/$2/RDFDAT
mv ZDNDAT ../data/TEST$1/$2/ZDNDAT
chmod 400 ../data/TEST$1/$2/*
```

which first creates a new DL_POLY *data/TEST..* sub-directory and then moves the standard DL_POLY output data files into it.

store requires two arguments:

store n a

where *n* is a unique string or number to label the output data in the *data/TESTn* sub-directory and *a* is a string: LF or VV according to the integration algorithm (leafrog - LF or velocity Verlet - VV).

Note that *store* sets the file access to read-only. This is to prevent the *store* macro overwriting existing data without your knowledge.

Bibliography

[1] Smith, W., and Forester, T., 1996, *J. Molec. Graphics*, **14**, 136. 3

[2] Smith, W., 1987, *Molecular Graphics*, **5**, 71. 3

[3] Sutton, A. P., and Chen, J., 1990, *Philos. Mag. Lett.*, **61**, 139. 4, 38, 87, 135

[4] Tersoff, J., 1989, *Phys. Rev. B*, **39**, 5566. 4, 34, 136

[5] van Gunsteren, W. F., and Berendsen, H. J. C. 1987, *Groningen Molecular Simulation (GROMOS) Library Manual.* BIOMOS, Nijenborgh, 9747 Ag Groningen, The Netherlands. Standard GROMOS reference. 4, 15

[6] Mayo, S., Olafson, B., and Goddard, W., 1990, *J. Phys. Chem.*, **94**, 8897. 4, 15, 33, 134

[7] Weiner, S. J., Kollman, P. A., Nguyen, D. T., and Case, D. A., 1986, *J. Comp. Chem.*, **7**, 230. 4, 15

[8] Smith, W., 2003, *Daresbury Laboratory.* 5, 11, 92, 100, 102, 119, 155, 162

[9] Smith, W., and Forester, T. R., 1994, *Comput. Phys. Commun.*, **79**, 52. 5

[10] Smith, W., and Forester, T. R., 1994, *Comput. Phys. Commun.*, **79**, 63. 5, 64

[11] Allen, M. P., and Tildesley, D. J. 1989, *Computer Simulation of Liquids.* Oxford: Clarendon Press. 5, 16, 50, 59, 62, 65, 84, 85

[12] Ryckaert, J. P., Ciccotti, G., and Berendsen, H. J. C., 1977, *J. Comput. Phys.*, **23**, 327. 5, 62, 84

[13] Andersen, H. C., 1983, *J. Comput. Phys.*, **52**, 24. 5, 64

[14] Fincham, D., 1992, *Molecular Simulation*, **8**, 165. 5, 60, 77

[15] Miller, T., Eleftheriou, M., Pattnaik, P., Ndirango, A., Newns, D., and Martyna, G., 2002, *J. Chem. Phys.*, **116**, 8649. 5, 62, 77

[16] Forester, T., and Smith, W., 1998, *J Computational Chemistry*, **19**, 102. 6, 60, 62, 79

[17] Martyna, G., Tuckerman, M., Tobias, D., and Klein, M., 1996, *Molecular Physics*, **87**, 1117. 6, 66, 78

[18] Evans, D. J., and Morriss, G. P., 1984, *Computer Physics Reports*, **1**, 297. 6, 60, 61, 65

[19] Berendsen, H. J. C., Postma, J. P. M., van Gunsteren, W., DiNola, A., and Haak, J. R., 1984, *J. Chem. Phys.*, **81**, 3684. 6, 60, 61, 65

[20] Hoover, W. G., 1985, *Phys. Rev.*, **A31**, 1695. 6, 60, 61, 65

[21] Jorgensen, W. L., Madura, J. D., and Swenson, C. J., 1984, *J. Amer. Chem. Soc*, **106**, 6638. 15

[22] Brode, S., and Ahlrichs, R., 1986, *Comput. Phys. Commun.*, **42**, 41. 17, 84, 85

[23] Hockney, R. W., and Eastwood, J. W. 1981, *Computer Simulation Using Particles*. McGraw-Hill International. 17, 87

[24] Vessal, B., 1994, *J. Non-Cryst. Solids*, **177**, 103. 20, 22, 33, 128, 134

[25] Smith, W., Greaves, G. N., and Gillan, M. J., 1995, *J. Chem. Phys.*, **103**, 3091. 20, 22, 33, 128, 134

[26] Smith, W., 1993, *CCP5 Information Quarterly*, **39**, 14. 22, 25, 28

[27] Clarke, J. H. R., Smith, W., and Woodcock, L. V., 1986, *J. Chem. Phys.*, **84**, 2290. 30, 31, 133

[28] Eastwood, J. W., Hockney, R. W., and Lawrence, D. N., 1980, *Comput. Phys. Commun.*, **19**, 215. 33, 36, 37

[29] Daw, M. S., and Baskes, M. I., 1984, *Phys. Rev. B*, **29**, 6443. 37, 135

[30] Foiles, S. M., Baskes, M. I., and Daw, M. S., 1986, *Chem. Phys. Lett.*, **33**, 7983. 37, 135

[31] Finnis, M. W., and Sinclair, J. E., 1984, *Philos. Mag. A*, **50**, 45. 37, 38, 135

[32] J., F., 1952, *Philos. Mag.*, **43**, 153. 37

[33] Rafii-Tabar, H., and Sutton, A. P., 1991, *Philos. Mag. Lett.*, **63**, 217. 38, 45, 135

[34] Todd, B., and Lynden-Bell, R., 1993, *Surf. Science*, **281**, 191. 38

[35] Cleri, F., and Rosato, F., 1993, *Phys. Rev. B*, **48**, 22. 38, 135

[36] Johnson, R. A., 1989, *Phys. Rev. B*, **39**, 12556. 45

[37] Smith, W., and Fincham, D., 1993, *Molecular Simulation*, **10**, 67. 51, 83, 84, 89, 104

[38] Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. G., 1995, *J. Chem. Phys.*, **103**, 8577. 52

[39] Hautman, J., and Klein, M. L., 1992, *Molecular Physics*, **75**, 379. 54, 55, 176

[40] Neumann, M., 1985, *J. Chem. Phys.*, **82**, 5663. 57

[41] Fincham, D., and Mitchell, P. J., 1993, *J. Phys. Condens. Matter*, **5**, 1031. 58

[42] Lindan, P. J. D., and Gillan, M. J., 1993, *J. Phys. Condens. Matter*, **5**, 1019. 59

[43] McCammon, J. A., and Harvey, S. C. 1987, *Dynamics of Proteins and Nucleic Acids*. Cambridge: University Press. 65

[44] Brown, D., and Clarke, J. H. R., 1984, *Molecular Physics*, **51**, 1243. 68

[45] Melchionna, S., Ciccotti, G., and Holian, B. L., 1993, *Molecular Physics*, **78**, 533. 69

[46] Tildesley, D. J., Streett, W. B., and Saville, G., l978, *Molec. Phys*, **35**, 639. 82

[47] Tildesley, D. J., and Streett, W. B. Multiple time step methods and an improved potential function for molecular dynamics simulations of molecular liquids. In Lykos, P., editor, *Computer Modelling of Matter*. ACS Symposium Series No. 86, 1978. 82

[48] Forester, T., and Smith, W., 1994, *Molecular Simulation*, **13**, 195. 82

[49] Smith, W., 1991, *Comput. Phys. Commun.*, **62**, 229. 83, 84, 88

[50] Smith, W., 1993, *Theoretica. Chim. Acta.*, **84**, 385. 83, 84, 85

[51] Smith, W., 1992, *Comput. Phys. Commun.*, **67**, 392. 84, 87

[52] Vessal, B., Amini, M., Leslie, M., and Catlow, C. R. A., 1990, *Molecular Simulation*, **5**, 1. 87

[53] Melchionna, S., and Cozzini, S., 1998, *University of Rome*. 101

Appendix A

The DL_POLY_2 Makefile

```
# Master makefile for DL_POLY_2.0
# Author: W. Smith December 2006
#
#=====================================================================
# Define default settings
#=====================================================================

BINROOT = ../execute
CC  = gcc
EX = DLPOLY.X
EXE = $(BINROOT)/$(EX)
FC=undefined
SHELL=/bin/sh
TYPE=par

#=====================================================================
# Define object files

OBJ_MOD = setup_module.o angles_module.o bonds_module.o \
        config_module.o core_shell_module.o dihed_module.o \
        ewald_module.o exclude_module.o external_field_module.o \
        four_body_module.o hkewald_module.o inversion_module.o \
        metal_module.o pmf_module.o property_module.o \
        rigid_body_module.o shake_module.o site_module.o \
        spme_module.otether_module.o three_body_module.o \
        vdw_module.o parse_module.o tersoff_module.o \
        pair_module.o

OBJ_ALL = angle_terms.o bond_terms.o core_shell_terms.o \
        coulomb_terms.o define_system.o dlpoly.o error.o \
```

```
        exclude_terms.o external_field_terms.o force_drivers.o \
        four_body_terms.o hkewald_terms.o inversion_terms.o \
        neu_coul_terms.o neu_ewald_terms.o nlist_builders.o \
        pmf_terms.o rigid_body_terms.o setup_program.o \
        shake_terms.o site_terms.o spme_terms.o strucopt.o \
        system_properties.o temp_scalers.o tether_terms.o \
        three_body_terms.o timchk.o traject.o utility_pack.o \
        warning.o tersoff_terms.o parse_tools.o ensemble_tools.o \
        kinetic_terms.o

OBJ_LF = lf_integrate.o lf_motion_1.o lf_rotation_1.o \
        lf_rotation_2.o pmf_lf.o

OBJ_VV = vv_integrate.o vv_motion_1.o vv_rotation_1.o \
        vv_rotation_2.o pmf_vv.o

OBJ_RRR = dihedral_terms.o ewald_terms.o metal_terms.o \
        vdw_terms.o

OBJ_PAR = basic_comms.o merge_tools.o pass_tools.o

#=================================================================
# Define targets
all:
        @echo "Error - please specify a target machine!"
        @echo "Permissible targets for this Makefile are:"
        @echo "                                              "
        @echo "gfortran                    (parallel)"
        @echo "hpcx                        (parallel)"
        @echo "crayxd1                     (parallel)"
        @echo "macosx-xlf-g5-mpi           (parallel)"
        @echo "hitachi-sr2201              (parallel)"
        @echo "sg8k-mpi                    (parallel)"
        @echo "         "
        @echo "Please examine Makefile for details"

# system specific targets follow :

#================== GNU Fortran, MPI version ====================
gfortran:
        $(MAKE) FC="mpif90" LD="mpif90 -o" \
        LDFLAGS="-O2 -ffast-math" \
        FFLAGS="-c -O2 -ffast-math" \
        EX=$(EX) BINROOT=$(BINROOT) $(TYPE)
```

```
#================== HPCx SP Power 5 ============================
hpcx:
        $(MAKE) FC="mpxlf" LD="mpxlf -o" \
        LDFLAGS="-O3 -q64 -qmaxmem=-1" \
        FFLAGS="-c -O3 -qmaxmem=-1 -qarch=pwr5 -qtune=pwr5 -qnosave" \
        EX=$(EX) BINROOT=$(BINROOT) $(TYPE)

#================= Cray XD1 (Portland Group) ===================
crayxd1:
        $(MAKE) LD="mpif90 -o" LDFLAGS="" \
        FC=mpif90 FFLAGS="-c -O -Mdalign" \
        EX=$(EX) BINROOT=$(BINROOT) $(TYPE)

#============= MacOSX-XLF-G5-MPI ================================
macosx-xlf-g5-mpi:
        $(MAKE) LD="/opt/ibmcmp/xlf/8.1/bin/xlf -o" \
        LDFLAGS="-L/opt/mpich-mx/lib -lmpich -lpmpich \
        -L/opt/mx/lib -lmyriexpress -L/usr/lib -lSystemStubs" \
        FC="/opt/ibmcmp/xlf/8.1/bin/xlf" \
        FFLAGS="-c -qstrict -O3 -qarch=auto -qmaxmem=32768"\
        EX=$(EX) BINROOT=$(BINROOT) $(TYPE)

#============ Hitachi SR2201 ====================================
hitachi-sr2201:
        $(MAKE) FC=xf90 \
        FFLAGS="-c -W0,'form(fixed),opt(o(3)),langlvl(save(0))' \
        -s,TRACE" intlist.o
        $(MAKE) LDFLAGS="" LDLIBS="-lfmpi -lmpi" LD="xf90 -o" \
        FC=xf90 \
        FFLAGS="-c -W0, 'form(fixed),opt(o(3)),langlvl(save(0))' \
        -s,TRACE" CC=xcc EX=$(EX) BINROOT=$(BINROOT) $(TYPE)

#=========== Silicon Graphics 8000 ==============================
sg8k-mpi:
        $(MAKE) LD="f90 -O3 -64 -o" FC=f90 LDFLAGS="-lmpi" \
        FFLAGS="-c -O3  -64" \
        EX=$(EX) BINROOT=$(BINROOT) $(TYPE)

#================================================================
# Default code for parallel (MPI) execution

par: check $(OBJ_MOD) $(OBJ_ALL) $(OBJ_RRR) $(OBJ_PAR)\
        $(OBJ_LF) $(OBJ_VV)
```

```
        $(LD) $(EX) $(OBJ_MOD) $(OBJ_ALL) $(OBJ_RRR) \
        $(OBJ_LF) $(OBJ_VV) $(OBJ_PAR)
        mv $(EX) $(EXE)

#================================================================
# Check that a machine has been specified
check:
        @if test $(FC) = "undefined";\
        then echo "You must specify a target machine!"; \
        exit 99;\
        fi

#================================================================
# Clean up the source directory
clean:
        rm -f $(OBJ_MOD) $(OBJ_ALL) $(OBJ_RRR) $(OBJ_PAR)\
        $(OBJ_LF) $(OBJ_VV) *.mod

#================================================================
# Declare dependencies
.f.o:
        $(FC) $(FFLAGS) $*.f
.c.o:
        $(CC) -c $*.c

#================================================================
# Declare dependency on module files

$(OBJ_ALL): $(OBJ_MOD)
$(OBJ_LF): $(OBJ_MOD)
$(OBJ_VV): $(OBJ_MOD)
$(OBJ_RRR): $(OBJ_MOD)
```

Appendix B

Periodic Boundary Conditions in DL_POLY

Introduction

DL_POLY_2 is designed to accommodate a number of different periodic boundary conditions, which are defined by the shape and size of the simulation cell. Briefly, these are as follows (which also indicates the IMCON flag defining the simulation cell type in the CONFIG File - see 4.1.2):

1. None e.g. isolated polymer in space. (IMCON=0).
2. Cubic periodic boundaries.(IMCON=1).
3. Orthorhombic periodic boundaries.(IMCON=2).
4. Parallelepiped periodic boundaries.(IMCON=3).
5. Truncated octahedral periodic boundaries. (IMCON=4).
6. Rhombic dodecahedral periodic boundaries. (IMCON=5).
7. Slab (X,Y periodic, Z nonperiodic). (IMCON=6).
8. Hexagonal prism periodic boundaries. (IMCON=7).

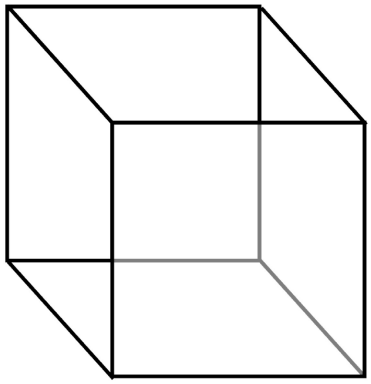
We shall now look at each of these in more detail. Note that in all cases the cell vectors and the positions of the atoms in the cell are to be specified in Angstroms (Å).

No periodic boundary (IMCON=0)

Simulations requiring no periodic boundaries are best suited to *in vacuuo* simulations, such as the conformational study of an isolated polymer molecule. This boundary condition is not recommended for studies in a solvent, since evaporation is likely to be a problem.

Note this boundary condition cannot be used with the Ewald summation method.

Cubic periodic boundaries (IMCON=1)



The cubic MD cell.

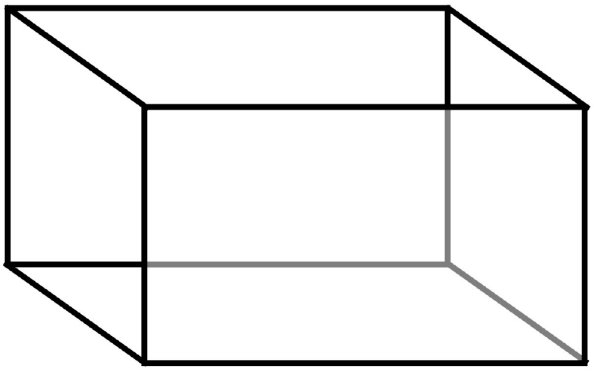
The cubic MD cell is perhaps the most commonly used in simulation and has the advantage of great simplicity. In DL_POLY_2 the cell is defined with the principle axes passing through the centres of the faces. Thus for a cube with sidelength D, the cell vectors appearing in the CONFIG file should be: (D,0,0); (0,D,0); (0,0,D). Note the origin of the atomic coordinates is the centre of the cell.

The cubic boundary condition can be used with the Ewald summation method.

Orthorhombic periodic boundaries (IMCON=2)

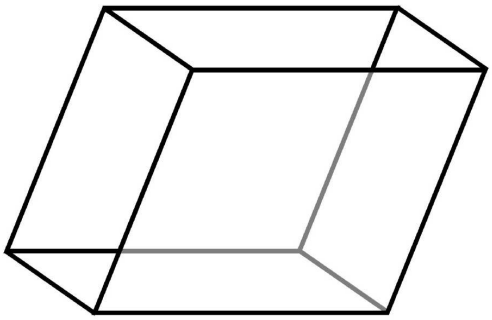
The orthorhombic cell is also a common periodic boundary, which closely resembles the cubic cell in use. In DL_POLY_2 the cell is defined with principle axes passing through the centres of the faces. For an orthorhombic cell with sidelengths D (in X-direction), E (in Y-direction) and F (in Z-direction), the cell vectors appearing in the CONFIG file should be: (D,0,0); (0,E,0); (0,0,F). Note the origin of the atomic coordinates is the centre of the cell.

The orthorhombic boundary condition can be used with the Ewald summation method.



The orthorhomic MD cell.

Parallelepiped periodic boundaries (IMCON=3)

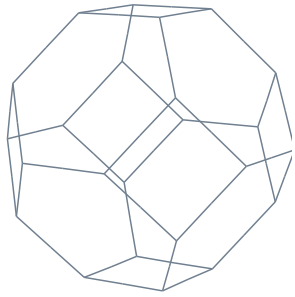


The parallelepiped MD cell.

The parallelepiped (e.g. monoclinic or triclinic) cell is generally used in simulations of crystalline materials, where its shape and dimension is commensurate with the unit cell of the crystal. Thus for a unit cell specified by three principal vectors $\underline{a}$, $\underline{b}$, $\underline{c}$, the MD cell is defined in the DL_POLY_2 CONFIG file by the vectors (La_1,La_2,La_3), (Mb_1,Mb_2,Mb_3), (Nc_1,Mc_2,Nc_3), in which L,M,N are integers, reflecting the multiplication of the unit cell in each principal direction. Note that the atomic coordinate origin is the centre of the MD cell.

The parallelepiped boundary condition can be used with the Ewald summation method.

Truncated octahedral boundaries (IMCON=4)



The truncated octahedral MD cell.

This is one of the more unusual MD cells available in DL_POLY, but it has the advantage of being more nearly spherical than most other MD cells. This means it can accommodate a larger spherical cutoff for a given number of atoms, which leads to greater efficiency. This can be very useful when simulating (for example) a large molecule in solution, where fewer solvent molecules are required for a given simulation cell width.

The principal axes of the truncated octahedron (see figure) pass through the centres of the square faces, and the width of the cell, measured from square face to square face along a principal axis defines the width D of the cell. From this, the cell vectors required in the DL_POLY_2 CONFIG file are simply: (D,0,0), (0,D,0), (0,0,D). These are also the cell vectors defining the enscribing cube, which posseses twice the volume of the truncated octahedral cell. Once again, the atomic positions are defined with respect to the cell centre.

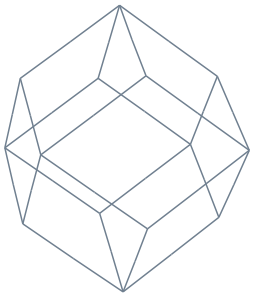
The truncated octahedron can be used with the Ewald summation method.

Rhombic dodecahedral boundaries (IMCON=5)

This is another unusual MD cell (see figure), but which possesses similar advantages to the truncated octahedron, but with a slightly greater efficiency in its use of the cell volume (the ratio is about 74% to 68%).

The principal axis in the X-direction of the rhombic dodecahedron passes through the centre of the cell and the centre of a rhombic face. The Y-axis does likewise, but is set at 90 degrees to the X-axis. The Z-axis completes the orthonormal set and passes through a vertex where four faces meet. If the width D of the cell is defined as the perpendicular distance between two opposite faces, the cell vectors required for the DL_POLY_2 CONFIG file are: (D,0,0), (0,D,0), (0,0,$\sqrt{2}$D).These also define the enscribing orthorhombic cell, which has twice the MD cell volume. In DL_POLY_2 the centre of the cell is also the origin of the atomic coordinates.

The rhombic dodecahedron can be used with the Ewald summation method.



The rhombic dodecahedral MD cell.

Slab boundary conditions (IMCON=6)

Slab boundaries are periodic in the X- and Y-directions, but not in the Z-direction. They are particularly useful for simulating surfaces. The periodic cell in the XY plane can be any parallelogram. The origin of the X,Y atomic coordinates lies on an axis perpendicular to the centre of the parallelogram. The origin of the Z coordinate is where the user specifies it, but at or near the surface is recommended.

If the XY parallelogram is defined by vectors $\underline{A}$ and $\underline{B}$, the vectors required in the CONFIG file are: (A_1,A_2,0), (B_1,B_2,0), (0,0,D), where D is any real number (including zero). If D is nonzero, it will be used by DL_POLY to help determine a 'working volume' for the system. This is needed to help calculate RDFs etc. (The working value of D is in fact taken as one of: 3×cutoff; or 2×max abs(Z coordinate)+cutoff; or the user specified D, whichever is the larger.)

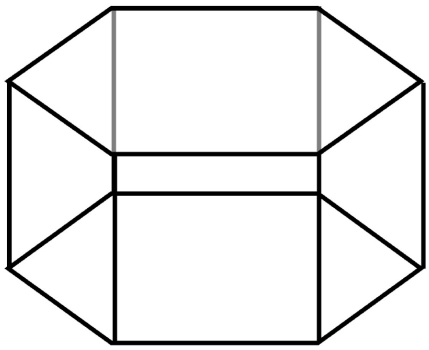
Note that the standard Ewald sum cannot be used with this boundary condition. DL_POLY_2 switches automatically to the Hautman-Klein-Ewald method instead [39].

The surface in a system with charges can also be modelled with DL_POLY_2 if periodicity is allowed in the Z-direction. In this case slabs of ions well-separated by vacuum zones in the Z-direction can be handled with IMCON=2 or 3.

Hexagonal prism boundaries (IMCON=7)

In this case the Z-axis lies along a line joining the centres of the hexagonal faces. The Y-axis is perpendicular to this and passes through the centre of one of the faces. The X-axis completes the orthonormal set and passes through the centre of an edge that is parallel to the Z-axis. (Note: It is important to get this convention right!) The origin of the atomic coordinates is the centre of the cell. If the length of one of the hexagon edges is D, the cell vectors required in the CONFIG file are: (3D,0,0), (0,$\sqrt{3}$D,0), (0,0,H), where H is the prism height (the distance between hexagonal faces). The orthorhombic cell also defined by these vectors enscribes the hexagonal prism and possesses twice the volume, but the height and the centre are the same.

The Ewald summation method may be used with this periodic boundary condition.



The hexagonal MD cell.

This MD cell is particularly suitable for simulating strands or fibres (i.e. systems with a pronounced anisotropy in the Z-direction), such as DNA strands in solution, or stretched polymer chains.

Appendix C

DL_POLY Error Messages and User Action

Introduction

In this appendix we document the error messages encoded in DL_POLY_2 and the recommended user action. The correct response is described as the **standard user response** in the approriate sections below, to which the user should refer before acting on the error encountered.

The reader should also be aware that some of the error messages listed below may be either disabled in, or absent from, the installed version of DL_POLY_2 . Disabled messages generally apply to older releases of the code, while absent messages apply to newer versions of the code and will not usually apply to previous releases. They are all included for completeness. Note that the wording of some of the messages may also have changed over time, usually to provide more specific information. The most recent wording appears below.

DL_POLY_2 incorporates FORTRAN 90 dynamic array allocation to set the array sizes at run time. It is not foolproof however. Sometimes an estimate of the required array sizes is difficult to obtain and the calculated value may be too small. For this reason DL_POLY_2 retains a number of array dimension checks and will terminate when an array bound error occurs.

When a dimension error occurs, the **standard user response** is to edit the DL_POLY_2 subroutine PARSET.F. Locate where the variable defining the array dimension is fixed and increase accordingly. To do this you should make use of the dimension information that DL_POLY_2 prints in the OUTPUT file prior to termination. If no information is supplied, simply doubling the size of the variable will usually do the trick. If the variable concerned is defined in one of the support subroutines CFGSCAN.F, FLDSCAN.F, CONSCAN.F you will need to insert a new line in PARSET.F to redefine it - after the relevant subroutine has been called! Finally the code must be recompiled, but in this case it will be necessary only to recompile PARSET.F and not the whole code.

The DL_POLY_2 Error Messages

Message 1: error - PVM_NODES unset

The code was C-preprocessed with the flag -DPVM set but the number of PVM nodes was not stated.

Action:
Delete the module INITCOMMS.O from the *srcf90* directory and re-make the executable,this time including the directive PVM_NODES=n (where n is the number of nodes you require) with the *make* command.

Message 2: error - machine not a hypercube

The number of nodes on the parallel machine is not a power of 2.

Action:
Specify an appropriate number of processors for job execution. If you are using PVM see *Action:* for error message 1.

Message 3: error - unknown directive found in CONTROL file

This error most likely arises when a directive is misspelt.

Action:
Locate incorrect directive in CONTROL file and replace.

Message 4: error - unknown directive found in FIELD file

This error most likely arises when a directive is misspelt or is encountered in an incorrect location in the FIELD file, which can happen if too few or too many data records are included.

Action:
Locate the erroneous directive in the FIELD file and correct error.

Message 5: error - unknown energy unit requested

The DL_POLY_2 FIELD file permits a choice of units for input of energy parameters. These may be: electron volts (**ev**); kilocalories (**kcal**); kilojoules (**kj**); or the DL_POLY_2 internal units (10 J mol^{-1}) (**internal**). There is no default value. Failure to specify any of these correctly, or reference to other energy units, will result in this error message. See documentation of the FIELD file.

Action:
Correct energy keyword on **units** directive in FIELD file and resubmit.

Message 6: error - energy unit not specified

A **units** directive is mandatory in the FIELD file. This error indicates that DL_POLY_2 has failed to find the required record.

Action:
Add **units** directive to FIELD file and resubmit.

Message 7: error - energy unit respecified

DL_POLY_2 expects only one **units** directive in the FIELD file. This error results if it encounters another - implying an ambiguity in units.

Action:
Locate extra **units** directive in FIELD file and remove.

Message 8: error - time step not specified

DL_POLY_2 requires a **timestep** directive in the CONTROL file. This error results if none is encountered.

Action:
Insert**timestep** directive in CONTROL file with an appropriate numerical value.

Message 10: error - too many molecule types specified

DL_POLY_2 has a set limit on the number of kinds of molecules it will handle in any simulation (this is not the same as the number of molecules). If this permitted maximum is exceeded, the program terminates. The error arises when the **molecules** directive in the FIELD file specifes too large a number.

Action:
Standard user response. Fix parameter `mxtmls`.

Message 11: error - duplicate molecule directive in FIELD file

The number of different types of molecules in a simulation should only be specified once. If DL_POLY_2 encounters more than one **molecules** directive, it will terminate execution.

Action:
Locate the extra **molecule** directive in the FIELD file and remove.

Message 12: error - unknown molecule directive in FIELD file

Once DL_POLY_2 encounters the **molecules** directive in the FIELD file, it assumes the following records will supply data describing the intramolecular force field. It does not then

expect to encounter directives not related to these data. This error message results if it encounters a unrelated directive. The most probable cause is incomplete specification of the data (e.g. when the **finish** directive has been omitted.)

Action:
Check the molecular data entries in the FIELD file and correct.

Message 13: error - molecule species not specified

This error arises when DL_POLY_2 encounters non-bonded force data in the FIELD file, *before* the molecular species have been specified. Under these circumstances it cannot assign the data correctly, and therefore terminates.

Action:
Make sure the molecular data appears before the non-bonded forces data in the FIELD file and resubmit.

Message 14: error - too many unique atom types specified

This error arises when DL_POLY_2 scans the FIELD file and discovers that there are too many different types of atoms in the system (i.e. the number of unique atom types exceeds the `mxsvdw` parameter.

Action:
Standard user response. Fix parameter `mxsvdw`.

Message 15: error - duplicate pair potential specified

In processing the FIELD file, DL_POLY_2 keeps a record of the specified short range pair potentials as they are read in. If it detects that a given pair potential has been specified before, no attempt at a resolution of the ambiguity is made and this error message results. See specification of FIELD file.

Action:
Locate the duplication in the FIELD file and rectify.

Message 16: error - strange exit from FIELD file processing

This should never happen! However one remote possibility is that there are more than 10,000 directives in the FIELD file! It simply means that DL_POLY_2 has ceased processing the FIELD data, but has not reached the end of the file or encountered a **close** directive. Probable cause: corruption of the DL_POLY_2 executable or of the FIELD file. We would be interested to hear of other reasons!

Action:
Recompile the program or recreate the FIELD file. If neither of these works, send the problem to us.

Message 17: error - strange exit from CONTROL file processing

See notes on message 16 above.

Message 18: error - duplicate 3-body potential specified

DL_POLY_2 has encountered a repeat specification of a 3-body potential in the FIELD file.

Action:
Locate the duplicate entry, remove and resubmit job.

Message 19: error - duplicate 4-body potential specified

A 4-body potential has been duplicated in the FIELD file.

Action:
Locate the duplicated 4-body potential and remove. Resubmit job.

Message 20: error - too many molecule sites specified

DL_POLY_2 has a fixed limit on the number of unique molecular sites in any given simulation. If this limit is exceeded, the program terminates.

Action:
Standard user response. Fix parameter `mxsite`.

Message 21: error - duplicate tersoff potential specified

The user has defined more than one Tersoff potential for a given pair of atoms types.
Action:
Locate the duplication in the FIELD file and correct.

Message 22: error - unsuitable radial increment in TABLE file

This arises when the tabulated potentials presented in the TABLE file have an increment that is greater than that used to define the other potentials in the simulation. Ideally the increment should be $r\_cut/(mxgrid-4)$, where r_cut is the potential cutoff for the short range potentials and `mxgrid` is the parameter defining the length of the interpolation arrays. An increment less than this is permissible however.

Action:
The tables must be recalculated with an appropriate increment.

Message 23: error - incompatible FIELD and TABLE file potentials

This error arises when the specification of the short range potentials is different in the FIELD and TABLE files. This usually means that the order of specification of the potentials is different. When DL_POLY_2 finds a change in the order of specification, it assumes that the user has forgotten to enter one.

Action:
Check the FIELD and TABLE files. Make sure that you correctly specify the pair potentials in the FIELD file, indicating which ones are to be presented in the TABLE file. Then check the TABLE file to make sure all the tabulated potentials are present in the order the FIELD file indicates.

Message 24: error - end of file encountered in TABLE file

This means the TABLE file is incomplete in some way: either by having too few potentials included, or the number of data points is incorrect.

Action:
Examine the TABLE file contents and regenerate it if it appears to be incomplete. If it look intact, check that the number of data points specified is what DL_POLY_2 is expecting.

Message 25: error - wrong atom type found in CONFIG file

On reading the input file CONFIG, DL_POLY_2 performs a check to ensure that the atoms specified in the configuration provided are compatible with the corresponding FIELD file. This message results if they are not.

Action:
The possibility exists that one or both of the CONFIG or FIELD files has incorrectly specified the atoms in the system. The user must locate the ambiguity, using the data printed in the OUTPUT file as a guide, and make the appropriate alteration.

Message 26: error - cutoff smaller than EAM potential range

DL_POLY_2 has detected an inconsistency in the definition of the EAM potential, namely that the user is not using the correct potential range.
Action:
Look up the correct range for this potential and adjust the DL_POLY cutoff accordingly.

Message 27: error - incompatible FIELD and TABEAM file potentials

The user has (or has not) specified a set of EAM potentials in the FIELD file which are not (or are) available in the TABEAM file.

Action:
Examine the FIELD file. Make sure you have correctly specified the EAM potentials. Check that these appear in the TABEAM file if required.

Message 28: error - transfer buffer too small in mettab

The number of points specifying an EAM potential in the TABEAM file exceeds the default buffer size in METTAB.F.
Action:
Reset the `mxbuff` parameter in PARSET.F subroutine to accommodate the required array length and recompile.

Message 29: error - end of file encountered in TABEAM file

DL_POLY_2 has reached the end of the TABEAM file without finding all the data it expects.
Action:
Either the TABEAM file is incomplete or it is improperly defined. Check the structure and content of the file with the TABEAM file specification in the manual and fix the error.

Message 30: error - too many chemical bonds specified

DL_POLY_2 sets a limit on the number of chemical bond potentials that can be specified in the FIELD file. Termination results if this number is exceeded. See FIELD file documentation. Do not confuse this error with that described by message 31 (below).

Action:
Standard user response. Fix parameter `mxtbnd`.

Message 31: error - too many chemical bonds in system

DL_POLY_2 sets a limit on the number of chemical bond potentials in the simulated system as a whole. (This number is a combination of the number of molecules and the number of bonds per molecule, divided by the number of processing nodes.) Termination results if this number is exceeded. Do not confuse this error with that described by message 30 (above).

Action:
Standard user response. Fix the parameter `mxbond`.

Message 32: error - integer array memory allocation failure

DL_POLY_2 has failed to allocate sufficient memory to accommodate one or more of the integer arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 33: error - real array memory allocation failure

DL_POLY_2 has failed to allocate sufficient memory to accommodate one or more of the real arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 34: error - character array memory allocation failure

DL_POLY_2 has failed to allocate sufficient memory to accommodate one or more of the character arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 35: error - logical array memory allocation failure

DL_POLY_2 has failed to allocate sufficient memory to accommodate one or more of the logical arrays in the code.

Action:
This may simply mean that your simulation is too large for the machine you are running on. Consider this before wasting time trying a fix. Try using more processing nodes if they are available. If this is not an option investigate the possibility of increasing the heap size for your application. Talk to your systems support people for advice on how to do this.

Message 36: error - failed fmet array allocation in mettab

DL_POLY_2 is unable to allocate the `fmet` array in the definition of an EAM potential.
Action:
Most probable cause is working too near the memory limit for the machine. Try using

more processors to free up some memory. Check the TABEAM file in case the data are incorrectly specified.

Message 40: error - too many bond constraints specified

DL_POLY_2 sets a limit on the number of bond constraints that can be specified in the FIELD file. Termination results if this number is exceeded. See FIELD file documentation. Do not confuse this error with that described by message 41 (below).

Action:
Standard user response. Fix the parameter `mxtcon`.

Message 41: error - too many bond constraints in system

DL_POLY_2 sets a limit on the number of bond constraints in the simulated system as a whole. (This number is a combination of the number of molecules and the number of per molecule, divided by the number of processing nodes.) Termination results if this number is exceeded. Do not confuse this error with that described by message 40 (above).

Action:
Standard user response. Fix the parameter `mxcons`.

Message 42: error - transfer buffer too small in merge1

The buffer used to transfer data between nodes in the MERGE1 subroutines has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 45: error - too many atoms in CONFIG file

DL_POLY_2 limits the number of atoms in the system to be simulated and checks for the violation of this condition when it reads the CONFIG file. Termination will result if the condition is violated.
Action:
Standard user response. Fix the parameter `mxatms`. Consider the possibility that the wrong CONFIG file is being used (e.g similar system, but larger size.)

Message 46: ewlbuf array too small in ewald1

The `ewlbuf` array used to store structure factor data in subroutine EWALD1 has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxebuf`.

Message 47: error - transfer buffer too small in merge

The buffer used to transfer data between nodes in the MERGE subroutines has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 48: error - transfer buffer too small in fortab

The buffer used to transfer data between nodes in the FORTAB subroutines has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 49: error - frozen core-shell unit specified

The DL_POLY_2 option to freeze the location of an atom (i.e. hold it permanently in one position) is not permitted for core-shell units. This includes freezing the core or the shell independently.

Action:
Remove the frozen atom option from the FIELD file. Consider using a non-polarisable atom instead.

Message 50: error - too many bond angles specified

DL_POLY_2 limits the number of valence angle potentials that can be specified in the FIELD file and checks for the violation of this. Termination will result if the condition is violated. Do not confuse this error with that described by message 51 (below).

Action:
Standard user response. Fix the parameter `mxtang`.

Message 51: error - too many bond angles in system

DL_POLY_2 limits the number of valence angle potentials in the system to be simulated (actually, the number to be processed by each node) and checks for the violation of this. Termination will result if the condition is violated. Do not confuse this error with that described by message 50 (above).

Action:
Standard user response. Fix the parameter `mxangl`. Consider the possibility that the wrong CONFIG file is being used (e.g similar system, but larger size.)

Message 52: error - end of FIELD file encountered

This message results when DL_POLY_2 reaches the end of the FIELD file, without having read all the data it expects. Probable causes: missing data or incorrect specification of integers on the various directives.

Action:
Check FIELD file for missing or incorrect data and correct.

Message 53: error - end of CONTROL file encountered

This message results when DL_POLY_2 reaches the end of the CONTROL file, without having read all the data it expects. Probable cause: missing **finish** directive.

Action:
Check CONTROL file and correct.

Message 54: error - problem reading CONFIG file

This message results when DL_POLY_2 encounters a problem reading the CONFIG file. Possible cause: corrupt data.

Action:
Check CONFIG file and correct.

Message 55: error - end of CONFIG file encountered

This error arises when DL_POLY_2 attempts to read more data from the CONFIG file than is actually present. The probable cause is an incorrect or absent CONFIG file, but it may be due to the FIELD file being incompatible in some way with the CONFIG file.

Action:
Check contents of CONFIG file. If you are convinced it is correct, check the FIELD file for inconsistencies.

Message 57: error - too many core-shell units specified

DL_POLY_2 has a restriction of the number of types of core-shell unit in the FIELD file and will terminate if too many are present. Do not confuse this error with that described by message 59 (below).

Action:
Standard user response. Fix the parameter `mxtshl`.

Message 59: error - too many core-shell units in system

DL_POLY_2 limits the number of core-shell units in the simulated system. Termination results if too many are encountered. Do not confuse this error with that described by message 57 (above).

Action:
Standard user response. Fix the parameter `mxshl`.

Message 60: error - too many dihedral angles specified

DL_POLY_2 will accept only a limited number of dihedral angles in the FIELD file and will terminate if too many are present. Do not confuse this error with that described by message 61 (below).

Action:
Standard user response. Fix the parameter `mxtdih`.

Message 61: error - too many dihedral angles in system

The number of dihedral angles in the whole simulated system is limited by DL_POLY_2 . Termination results if too many are encountered. Do not confuse this error with that described by message 60 (above).

Action:
Standard user response. Fix the parameter `mxdihd`.

Message 62: error - too many tethered atoms specified

DL_POLY_2 will accept only a limited number of tethered atoms in the FIELD file and will terminate if too many are present. Do not confuse this error with that described by message 63 (below).

Action:
Standard user response. Fix the parameter `mxteth`.

Message 63: error - too many tethered atoms in system

The number of tethered atoms in the simulated system is limited by DL_POLY_2 . Termination results if too many are encountered. Do not confuse this error with that described by message 62 (above).

Action:
Standard user response. Fix the parameter `msteth`.

Message 65: error - too many excluded pairs specified

This error can arise when DL_POLY_2 is identifying the atom pairs that cannot have a pair potential between them, by virtue of being chemically bonded for example (see subroutine EXCLUDE). Some of the working arrays used in this operation may be exceeded, resulting in termination of the program.

Action:
Standard user response. Fix the parameter `mxexcl`.

Message 66: error - incorrect boundary condition for HK ewald

The Hautman-Klein Ewald method can only be used with XY planar periodic boundary conditions (i.e. `imcon` = 6).

Action:
Either the periodic boundary condition, or the choice of calculation of the electrostatic forces must be changed.

Message 67: error - incorrect boundary condition in thbfrc

Three body forces in DL_POLY_2 are only permissible with cubic , orthorhombic and parallelepiped periodic boundaries. Use of other boundary conditions results in this error.

Action:
If nonperiodic boundaries are required, the only option is to use a very large simulation cell, with the required system at the centre surrounded by a vacuum. This is not very efficient however and use of a realistic periodic system is the best option.

Message 69: error - too many link cells required in thbfrc

The calculation of three body forces in DL_POLY_2 is handled by the link cell algorithm. This error arises if the required number of link cells exceeds the permitted array dimension in the code.

Action:
Standard user response. Fix the parameter `mxcell`.

Message 70: error - constraint bond quench failure

When a simulation with bond constraints is started, DL_POLY_2 attempts to extract the kinetic energy of the constrained atom-atom bonds arising from the assignment of initial random velocities. If this procedure fails, the program will terminate. The likely cause is a badly generated initial configuration.

Action:
Some help may be gained from increasing the cycle limit, by following the standard user response to increase the control parameter `mxshak`. You may also consider reducing the tolerance of the SHAKE iteration, the directive **shake** in the CONTROL file. However it is probably better to take a good look at the starting conditions!

Message 71: error - too many metal potentials specified

The number of metal potentials that can be specfied in the FIELD file is limited. This error results if too many are used.

Action:
Standard user response. Fix the parameter `mxvdw`. Note that this parameter must be *double* the number of required metal potentials. Recompile the program.

Message 72: error - different metal potential types specified

DL_POLY_2 does not permit the user to mix different types of metal potential in the same simulation. There are no known rules for making alloys in this way.
Action:
Change the FIELD (and TABEAM) file as required so that only one type of metal potential is used.

Message 73: error - too many inversion potentials specified

The number of inversion potentials specified in the FIELD file exceeds the permitted maximum.

Action:
Standard user response. Fix the parameter `mxtinv`.

Message 75: error - too many atoms in specified system

DL_POLY_2 places a limit on the number of atoms that can be simulated. Termination results if too many are specified.

Action:
Standard user response. Fix the parameter `mxatms`.

Message 77: error - too many inversion potentials in system

The simulation contains too many inversion potentials overall, causing termination of run.

Action:
Standard user response. Fix the parameter `mxinv`.

Message 79: error - incorrect boundary condition in fbpfrc

The 4-body force routine assumes a cubic or parallelepiped periodic boundary condition is in operation. The job will terminate if this is not adhered to.

Action:
You must reconfigure your simulation to an appropriate boundary condition.

Message 80: error - too many pair potentials specified

DL_POLY_2 places a limit on the number of pair potentials that can be specified in the FIELD file. Exceeding this number results in termination of the program execution.

Action:
Standard user response. Fix the parameters `mxsvdw`. and `mxvdw`.

Message 81: error - unidentified atom in pair potential list

DL_POLY_2 checks all the pair potentials specified in the FIELD file and terminates the program if it can't identify any one of them from the atom types specified earlier in the file.

Action:
Correct the erroneous entry in the FIELD file and resubmit.

Message 82: error - calculated pair potential index too large

In checking the pair potentials specified in the FIELD file DL_POLY_2 calculates a unique integer index that henceforth identifies the potential within the program. If this index becomes too large, termination of the program results.

Action:
Standard user response. Fix the parameters `mxsvdw` and `mxvdw`.

Message 83: error - too many three body potentials specified

DL_POLY_2 has a limit on the number of three body potentials that can be defined in the FIELD file. This error results if too many are included.

Action:
Standard user response. Fix the parameter `mxtbp`.

Message 84: error - unidentified atom in 3-body potential list

DL_POLY_2 checks all the 3-body potentials specified in the FIELD file and terminates the program if it can't identify any one of them from the atom types specified earlier in the file.

Action:
Correct the erroneous entry in the FIELD file and resubmit.

Message 85: error - required velocities not in CONFIG file

If the user attempts to start up a DL_POLY_2 simulation with the **restart** or **restart scale** directives (see description of CONTROL file,) the program will expect the CONFIG file to contain atomic velocities as well as positions. Termination results if these are not present.

Action:
Either replace the CONFIG file with one containing the velocities, or if not available, remove the **restart** directive altogether and let DL_POLY_2 create the velocities for itself.

Message 86: error - calculated 3-body potential index too large

DL_POLY_2 has a permitted maximum for the calculated index for any three body potential in the system (i.e. as defined in the FIELD file). If there are m distinct types of atom in the system, the index can possibly range from 1 to $(m^2 * (m - 1))/2$. If the internally calculated index exceeds this number, this error report results.

Action:
Standard user response. Fix the parameter `mxtbp`.

Message 87: error - too many link cells required in fbpfrc

The FBPFRC subroutine uses link cells to compute the four body forces. This message indicates that the link cell arrays have insufficient size to work properly.

Action:
Standard user response. Fix the parameter `mxcell`.

Message 88: error - too many tersoff potentials specified

Too many Tersoff potentials have been defined in the FIELD file. Certain arrays must be increased in size to accommodate the data.

Action:
Standard user response. Fix the parameter `mxter`.

Message 89: error - too many four body potentials specified

Too many four body potential have been defined in the FIELD file. Certain arrays must be increased in size to accommodate the data.

Action:
Standard user response. Fix the parameter `mxfbp`.

Message 90: error - system total electric charge nonzero

In DL_POLY_2 a check on the total system charge will result in an error if the net charge of the system is nonzero. (Note: In DL_POLY_2 this message has been disabled. The program merely prints a warning stating that the system is not electrically neutral but it does not terminate the program - watch out for this.)

Action:
Check the specified atomic charges and their populations. Make sure they add up to zero. If the system is required to have a net zero charge, you can enable the call to this error message in subroutine SYSDEF.

Message 92: error - unidentified atom in tersoff potential list

The specification of a Tersoff potential in the FIELD file has referenced an atom type that is unknown.

Action:
Locate the erroneous atom type in the Tersoff potential definition in the FIELD file and correct. Make sure this atom type is specified by an `atoms` directive earlier in the file.

Message 91: error - unidentified atom in 4-body potential list

The specification of a four-body potential in the FIELD file has referenced an atom type that is unknown.

Action:
Locate the erroneous atom type in the four body potential definition in the FIELD file and correct. Make sure this atom type is specified by an `atoms` directive earlier in the file.

Message 93: error - cannot use shell model with rigid molecules

The dynamical shell model implemented in DL_POLY_2 is not designed to work with rigid molecules. This error results if these two options are simultaneously selected.

Action:
In some circumstances you may consider overriding this error message and continuing with your simulation. For example if your simulation does not require the polarisability to be a feature of the rigid species, but is confined to free atoms or flexible molecules in the same system. The appropriate error trap is found in subroutine SYSDEF.

Message 95: error - potential cutoff exceeds half cell width

In order for the minimum image convention to work correctly within DL_POLY_2 , it is necessary to ensure that the cutoff applied to the pair potentials does not exceed half

the perpendicular width of the simulation cell. (The perpendicular width is the shortest distance between opposing cell faces.) Termination results if this is detected. In NVE simulations this can only happen at the start of a simulation, but in NPT, it may occur at any time.

Action:
Supply a cutoff that is less than half the cell width. If running constant pressure calculations, use a cutoff that will accommodate the fluctuations in the simulation cell. Study the fluctuations in the OUTPUT file to help you with this.

Message 97: error - cannot use shell model with neutral groups

The dynamical shell model was not designed to work with neutral groups. This error results if an attempt is made to combine both.

Action:
There is no general remedy for this error if you wish to combine both these capabilities. However if your simulation does not require the polarisability to be a feature of rigid species (comprising the charged groups), but is confined to free atoms or flexible molecules in the same system, you may consider overriding this error message and continuing with your simulation. The appropriate error trap is found in subroutine SYSDEF.

Message 99: error - cannot use shell model with constraints

The dynamical shell model was not designed to work in conjunction with constraint bonds. This error results if both are used in the same simulation.

Action: There is no general remedy if you wish to combine both these capabilities. However if your simulation does not require the polarisability to be a feature of the constrained species, but is confined to free atoms or flexible molecules, you may consider overriding this error message and continuing with your simulation. The appropriate error trap is in subroutine SYSDEF.

Message 100: error - forces working arrays too small

There are a number of arrays in DL_POLY_2 that function as workspace for the forces calculations. Their dimension is equal to the number of atoms in the simulation cell divided by the number of nodes. If these arrays are likely to be exceeded, DL_POLY_2 will terminate execution.

Action:
Standard user response. Fix the parameter `msatms`.

Message 101: error - calculated 4-body potential index too large

DL_POLY_2 has a permitted maximum for the calculated index for any four body potential in the system (i.e. as defined in the FIELD file). If there are m distinct types of atom in the system, the index can possibly range from 1 to $(m^2 * (m + 1) * (m + 2))/6$. If the internally calculated index exceeds this number, this error report results.

Action:
Standard user response. Fix the parameter `mxfbp`.

Message 102: error - parameter mxproc exceeded in shake arrays

The RD-SHAKE algorithm distributes data over all nodes of a parallel computer. Certain arrays in RD-SHAKE have a minimum dimension equal to the maximum number of nodes DL_POLY_2 is likely to encounter. If the actual number of nodes exceeds this, the program terminates.

Action:
Standard user response. Fix the parameter `mxproc`.

Message 103: error - parameter mxlshp exceeded in shake arrays

The RD-SHAKE algorithm requires that information about 'shared' atoms be passed between nodes. If there are too many atoms, the arrays holding the information will be exceeded and DL_POLY_2 will terminate execution.

Action:
Standard user response. Fix the parameter `mxlshp`.

Message 105: error - shake algorithm failed to converge

The RD-SHAKE algorithm for bond constraints is iterative. If the maximum number of permitted iterations is exceeded, the program terminates. Possible causes include: a bad starting configuration; too large a time step used; incorrect force field specification; too high a temperature; inconsistent constraints involving shared atoms etc.

Action:
Corrective action depends on the cause. It is unlikely that simply increasing the iteration number will cure the problem, but you can try: follow the standard user response to increase the control parameter `mxshak`. But the trouble is much more likely to be cured by careful consideration of the physical system being simulated. For example, is the system stressed in some way? Too far from equilibrium?

Message 106: error - neighbour list array too small in parlink

Construction of the Verlet neighbour list in subroutine `parlink` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 107: error - neighbour list array too small in parlinkneu

Construction of the Verlet neighbour list in subroutine `parlinkneu` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 108: error - neighbour list array too small in parneulst

Construction of the Verlet neighbour list in subroutine `parneulst` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 109: error - neighbour list array too small in parlst_nsq

Construction of the Verlet neighbour list in subroutine `parlst_nsq` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 110: error - neighbour list array too small in parlst

Construction of the Verlet neighbour list in subroutine `parlst` nonbonded (pair) force has exceeded the neighbour list array dimensions.

Action:
Standard user response. Fix the parameter `mxlist`.

Message 112: error - vertest array too small

This error results when the dimension of the DL_POLY_2 VERTEST arrays, which are used in checking if the Verlet list needs updating, have been exceeded.

Action:
Standard user response. Fix the parameter `mslst`.

Message 120: error - invalid determinant in matrix inversion

DL_POLY_2 occasionally needs to calculate matrix inverses (usually the inverse of the matrix of cell vectors, which is of size 3×3). For safety's sake a check on the determinant is made, to prevent inadvertent use of a singular matrix.

Action:
Locate the incorrect matrix and fix it - e.g. are cell vectors correct?

Message 130: error - incorrect octahedral boundary condition

When calculating minimum images DL_POLY_2 checks that the periodic boundary of the simulation cell is compatible with the specifed minimum image algorithm. Program termination results if an inconsistency is found. In this case the error refers to the truncated octahedral minimum image, which is inconsistent with the simulation cell. The most probable cause is the incorrect definition of the simulation cell vectors present in the input file CONFIG, these must equal the vectors of the enscribing cubic cell.

Action:
Check the specified simulation cell vectors and correct accordingly.

Message 135: error - incorrect hexagonal prism boundary condition

When calculating minimum images DL_POLY_2 checks that the periodic boundary of the simulation cell is compatible with the specifed minimum image algorithm. Program termination results if an inconsistency is found. In this case the error refers to the hexagonal prism minimum image, which is inconsistent with the simulation cell. The most probable cause is the incorrect definition of the simulation cell vectors present in the input file CONFIG, these must equal the vectors of the enscribing orthorhombic cell.

Action:
Check the specified simulation cell vectors and correct accordingly.

Message 140: error - incorrect dodecahedral boundary condition

When calculating minimum images DL_POLY_2 checks that the periodic boundary of the simulation cell is compatible with the specifed minimum image algorithm. Program termination results if an inconsistency is found. In this case the error refers to the rhombic dodecahedral minimum image, which is inconsistent with the simulation cell. The most probable cause is the incorrect definition of the simulation cell vectors present in the input file CONFIG, these must equal the vectors of the enscribing tetragonal simulation cell.

Action:
Check the specified simulation cell vectors and correct accordingly.

Message 141: error - duplicate metal potential specified

The user has specified a particular metal potential more than once in the FIELD file.

Action:
Locate the metal potential specification in the FIELD file and remove or correct the potential concerned.

Message 145: error - no van der waals potentials defined

This error arises when there are no VDW potentials specified in the FIELD file but the user has not specified **no vdw** in the CONTROL file. In other words DL_POLY_2 expects the FIELD file to contain VDW potential specifications.

Action:
Edit the FIELD file to insert the required potentials or specify **no vdw** in the CONTROL file.

Message 150: error - unknown van der waals potential selected

DL_POLY_2 checks when constructing the interpolation tables for the short ranged potentials that the potential function requested is one which is of a form known to the program. If the requested potential form is unknown, termination of the program results. The most probable cause of this is the incorrect choice of the potential keyword in the FIELD file or one in the wrong columns (input is formatted).

Action:
Read the DL_POLY_2 documentation and find the potential keyword for the potential desired. Insert the correct index in the FIELD file definition and ensure it occurs in the correct columns (17-20). If the correct form is not available, look at the subroutine FORGEN (or its variant) and define the potential for yourself. It is easily done.

Message 151: error - unknown metal potential selected

The metal potentials available in DL_POLY_2 are confined to density dependent forms of the Sutton-Chen type. This error results if the user attempts to specify another.

Action:
Re-specify the potential as Sutton-Chen type if possible. Check the potential keyword appears in columns 17-20 of the FIELD file.

Message 153: error - metals not permitted with multiple timestep

The multiple timestep algorithm cannot be used in conjunction with metal potentials in DL_POLY_2 .

Action:
The simulation must be run without the multiple timestep option.

Message 160: error - unaccounted for atoms in exclude list

This error message means that DL_POLY_2 has been unable to find all the atoms described in the exclusion list within the simulation cell. This should never occur, if it does it means a serious bookkeeping error has occured. The probable cause is corruption of the code somehow.

Action:
If you feel you can tackle it - good luck! Otherwise we recommend you get in touch with the program authors. Keep all relevant data files to help them find the problem.

Message 170: error - too many variables for statistic array

This error means the statistics arrays appearing in subroutine STATIC are too small. This can happen if the number of unique atom types is too large.

Action:
Standard user response. Fix the parameter `mxnstk`. `mxnstk` should be at least (45+*number of unique atom types*).

Message 180: error - Ewald sum requested in non-periodic system

DL_POLY_2 can use either the Ewald method or direct summation to calculate the electrostatic potentials and forces in periodic (or pseudo-periodic) systems. For non-periodic systems only direct summation is possible. If the Ewald summation is requested (with the **ewald sum** or **ewald precision** directives in the CONTROL file) without periodic boundary conditions, termination of the program results.

Action:
Select periodic boundaries by setting the variable `imcon`>0 in the CONFIG file (if possible) or use a different method to evaluate electrostatic interactions e.g. by usinf the **coul** directive in the CONTROL file.

Message 185: error - too many reciprocal space vectors

DL_POLY_2 places hard limit on the number of k vectors to be used in the Ewald sum and terminates if more than this is requested.

Action:
Either consider using fewer k vectors in the Ewald sum (and a larger cutoff in real space) or follow standard user response to reset the parameters `kmaxb, kmaxc`.

Message 186: error - transfer buffer array too small in sysgen

In the subroutine SYSGEN.F DL_POLY_2 requires dimension of the array `buffer` (defined by the parameter `mxbuff`) to be no less than the parameter `mxatms` or the product of parameters `mxnstk*mxstak`. If this is not the case it will be unable to restart the program correctly to continue a run. (Applies to parallel implementations only.)

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 190: error - buffer array too small in splice

DL_POLY_2 uses a workspace array named `buffer` in several routines. Its declared size is a compromise of several rôles and may sometimes be too small (though in the supplied program, this should happen only very rarely). The point of failure is in the SPLICE routine, which is part of the RD-SHAKE algorithm.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 200: error - rdf buffer array too small in revive

This error indicates that the buffer array used to globally sum the rdf arrays in subroutine REVIVE is too small.

Action:
Standard user response. Fix the parameter `mxbuff`. Alternatively `mxrdf` can be set smaller.

Message 220: error - too many neutral groups in system

DL_POLY_2 has a fixed limit on the number of charged groups in a simulation. This error results if the number is exceeded.

Action:
Standard user response. Fix the parameter `mxneut`.

Message 230: error - neutral groups improperly arranged

In the DL_POLY_2 FIELD file the charged groups must be defined in consecutive order. This error results if this convention is not adhered to.

Action:
The arrangement of the data in the FIELD file must be sorted. All atoms in the same group must be arranged consecutively. Note that reordering the file in this way implies a rearrangement of the CONFIG file also.

Message 250: error - Ewald sum requested with neutral groups

DL_POLY_2 will not permit the use of neutral groups with the Ewald sum. This error results if the two are used together.

Action:
Either remove the **neut** directive from the FIELD file or use a different method to evaluate the electrostatic interactions.

Message 260: error - parameter mxexcl exceeded in excludeneu routine

An error has been detected in the construction of the excluded atoms list for neutral groups. This occurs when the parameter `mxexcl` is exceeded in the EXCLUDENEU routine.

Action:
Standard user response. Fix parameter `mxexcl`.

Message 300: error - incorrect boundary condition in parlink

The use of link cells in DL_POLY_2 implies the use of appropriate boundary conditions. This error results if the user specifies octahedral, dodecahedral or slab boundary conditions.

Action:
The simulation must be run with cubic, orthorhombic or parallelepiped boundary conditions.

Message 301: error - too many rigid body types

The maximum number of rigid body types permitted by DL_POLY_2 has been exceeded.

Action:
Standard user response. Fix the parameter `mxungp`.

Message 302: error - too many sites in rigid body

This error arises when DL_POLY_2 finds that the number of sites in a rigid body exceeds the dimensions of the approriate storage arrays.

Action:
Standard user response. Fix the parameter `mxngp`.

Message 303: error - too many rigid bodies specified

The maximum number of rigid bodies in a simulation has been reached. Do not confuse this with message 304 below.

Action:
Standard user response. Fix the parameter `mxgrp`.

Message 304: error - too many rigid body sites in system

This error occurs when the total number of sites within all rigid bodies exceeds the permitted maximum. Do not confuse this with message 303 above.

Action:
Standard user response. Fix the parameter `mxgatms`.

Message 305: error - box size too small for link cells

The link cells algorithm in DL_POLY_2 cannot work with less than 27 link cells. Depending on the cell size and the chosen cut-off, DL_POLY_2 may decide that this minimum cannot be achieved and terminate.

Action:
If a smaller cut-off is acceptable use it. Otherwise do not use link cells. Consider running a larger system, where link cells will work.

Message 306: error - failed to find principal axis system

This error indicates that the routine QUATBOOK has failed to find the principal axis for a rigid unit.

Action:
This is an unlikely error. The code should correctly handle linear, planar and 3-dimensional rigid units. Check the definition of the rigid unit in the CONFIG file, if sensible report the error to the authors.

Message 310: error - quaternion setup failed

This error indicates that the routine QUATBOOK has failed to reproduce all the atomic positions in rigid units from the centre of mass and quaternion vectors it has calculated.

Action:
Check the contents of the CONFIG file. DL_POLY_2 builds its local body description of a rigid unit type from the *first* occurrence of such a unit in the CONFIG file. The error most likely occurs because subsequent occurrences were not sufficiently similar to this reference

structure. If the problem persists increase the value of the variable `tol` in QUATBOOK and recompile. If problems still persist double the value of `dettest` in QUATBOOK and recompile. If you still encounter problems contact the authors.

Message 320: error - site in multiple rigid bodies

DL_POLY_2 has detected that a site is shared by two or more rigid bodies. There is no integration algorithm available in this version of the package to deal with this type of model.

Action:
The only course is to redefine the molecular model (e.g. introducing flexible bonds and angles in suitable places) to allow DL_POLY_2 to proceed.

Message 321: error - quaternion integrator failed

The quaternion algorithm has failed to converge. If the maximum number of permitted iterations is exceeded, the program terminates. Possible causes include: a bad starting configuration; too large a time step used; incorrect force field specification; too high a temperature; inconsistent constraints involving shared atoms etc.

Action:
Corrective action depends on the cause. Try reducing the timestep or running a zero kelvin structure optimization for a hundred timesteps or so. It is unlikely that simply increasing the iteration number will cure the problem, but you can try: follow the standard user response to increase the parameter `mxquat`. But the trouble is much more likely to be cured by careful consideration of the physical system being simulated. For example, is the system stressed in some way? Too far from equilibrium?

Message 330: error - mxewld parameter incorrect

DL_POLY_2 has two strategies for parallelization of the reciprocal space part of the Ewald sum. If EWALD1 is used the parameter `mxewld` should equal the parameter `msatms`. If EWALD1A is used this parameter should equal `mxatms`.

Action:
Standard user response. Set the parameter `mxewld` to the value appropriate for the version of EWALD1 you are using. Recompile the program.

Message 331: error - mxhke parameter incorrect

The parameter `mxhke`, which defines the dimension of some arrays used in the Hautman-Klein Ewald method, should equal the parameter `msatms`.

Action:
Standard user response. Set the parameter `mxhke` to the value regquired. Recompile the program.

Message 332: error - mxhko parameter too small

The parameter `mxhko` defines the maximum order for the Taylor expansion implicit in the Hautman-Klein Ewald method. DL_POLY_2 has a maximum of `mxhko` = 3, but it can be set to less in some implementations. If this error arises when the user requestes an order in excess of this parameter.

Action:
Standard user response. Set the parameter `mxhko` to a higher value (if it is <3) and recompile the program. Alternatively request a lower order in the CONTROL file through the `nhko` variable (see 4.1.1).

Message 340: error - invalid integration option requested

DL_POLY_2 has detected an incompatibility in the simulation instructions, namely that the requested integration algorithm is not compatible with the physical model. It *may* be possible to override this error trap, but it is up to the user to establish if this is sensible.
Action:
This is a non recoverable error, unless the user chooses to override the restriction.

Message 350: error - too few degrees of freedom

This error can arise if a small system is being simulated and the number of constraints applied is too large.

Action:
Simulate a larger system or reduce the number of constraints.

Message 360: error - frozen atom found in rigid body

DL_POLY_2 does not permit a site in a rigid body to be frozen i.e. fixed in one location in space.

Action:
Remove the 'freeze' condition from the site concerned. Consider using a very high site mass to achieve a similar effect.

Message 380: error - simulation temperature not specified

DL_POLY_2 has failed to find a **temp** directive in the CONTROL file.

Action:
Place a **temp** directive in the CONTROL file, with the required temperature specified.

Message 381: error - simulation timestep not specified

DL_POLY_2 has failed to find a **timestep** directive in the CONTROL file.

Action:
Place a **timestep** directive in the CONTROL file, with the required timestep specified.

Message 382: error - simulation cutoff not specified

DL_POLY_2 has failed to find a **cutoff** directive in the CONTROL file.

Action:
Place a **cutoff** directive in the CONTROL file, with the required forces cutoff specified.

Message 383: error - simulation forces option not specified

DL_POLY_2 has failed to find any directive specifying the electrostatic interactions options in the CONTROL file.

Action:
Ensure the CONTROL file contains at least one directive specifying the electrostatic potentials (e.g. **ewald**, **coul**, **no electrostatics** etc.)

Message 384: error - verlet strip width not specified

DL_POLY_2 has failed to find the **delr** directive in the CONTROL file.

Action:
Insert a **delr** directive in the CONTROL file, specifying the width of the verlet strip augmenting the forces cutoff.

Message 385: error - primary cutoff not specified

DL_POLY_2 has failed to find the **prim** directive in the CONTROL file. Necessary only if multiple timestep option required.

Action:
Insert a **prim** directive in the CONTROL file, specifying the primary cutoff radius in the multiple timestep algorithm.

Message 386: error - primary cutoff larger than rcut

The primary cutoff specified by the **prim** directive in the CONTROL file exceeds the value specified for the forces cutoff (directive **cut**). Applies only if the multiple timestep option is required.

Action:
Locate the **prim** directive in the CONTROL file, and alter the chosen cutoff. Alternatively, increase the real space cutoff specified with the **cut** directive. Take care to avoid error number 398.

Message 387: error - system pressure not specified

The target system pressure has not been specified in the CONTROL file. Applies to NPT simulations only.

Action:
Insert a **press** directive in the CONTROL file specifying the required system pressure.

Message 388: error - npt incompatible with multiple timestep

The use of NPT (constant pressure) and temperature is not compatible with the multiple timestep option.

Action:
Simulation must be run at fixed volume in this case. But note it may be possible to use NPT without the multiple timestep, in ourder to estimate the required system volume, then switch back to multiple timestep and NVT dynamics at the required volume.

Message 389: number of pimd beads not specified in field file

The user has failed to specify how many quantum beads is required in a Path Integral Molecular Dyamics simulation. Applies to PIMD version of DL_POLY_2 only.

Action:
The required numer of beads must be specified in the FIELD file.

Message 390: error - npt ensemble requested in non-periodic system

A non-periodic system has no defined volume, hence the NPT algorithm cannot be applied.

Action:
Either simulate the system with a periodic boundary, or use another ensemble.

Message 391: incorrect number of pimd beads in config file

The CONFIG file must specify the position of all the beads in a PIMD simulation, not just the positions of the parent atoms, otherwise this error results.

Action:
The CONFIG file must be reconstructed to provide the required data.

Message 392: error - too many link cells requested

The number of link cells required for a given simulation exceeds the number allowed for by the DL_POLY_2 arrays.

Action:
Standard user response. Fix the parameter `mxcell`.

Message 394: error - minimum image arrays exceeded

The work arrays used in IMAGES have been exceeded.
Action: Standard user response. Fix the parameter `mxxdf`.

Message 396: error - interpolation array exceeded

DL_POLY_2 has sought to read past the end of an interpolation array. This should never happen!

Action:
Contact the authors.

Message 398: error - cutoff too small for rprim and delr

This error can arise when the multiple timestep option is used. It is essential that the primary cutoff (`rprim`) is less than the real space cutoff (`rcut`) by at least the Verlet shell width `delr` (preferably much larger!). DL_POLY_2 terminates the run if this condition is not satisfied.

Action:
Adjust `rcut`, `rprim` and `delr` to satisfy the DL_POLY_2 requirement. These are defined with the directives **cut**, **prim** and **delr** respectively.

Message 400: error - rvdw greater than cutoff

DL_POLY_2 requires the real space cutoff (`rcut`) to be larger than, or equal to, the van der Waals cutoff (`rvdw`) and terminates the run if this condition is not satisfied.

Action:
Adjust `rvdw` and `rcut` to satisfy the DL_POLY_2 requirement.

Message 402: error - van der waals cutoff unset

The user has not set a cutoff (`rvdw`) for the van der Waals potentials. The simulation cannot proceed without this being specified.

Action:
Supply a cutoff value for the van der Waals terms in the CONTROL file using the directive **rvdw**, and resubmit job.

Message 410: error - cell not consistent with image convention

The simulation cell vectors appearing in the CONFIG file are not consistent with the specified image convention.

Action:
Locate the variable `imcon` in the CONFIG file and correct to suit the cell vectors.

Message 412: error - mxxdf parameter too small for shake routine

In DL_POLY_2 the parameter `mxxdf` must be greater than or equal to the parameter `mxcons`. If it is not, this error is a possible result.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 414: error - conflicting ensemble options in CONTROL file

DL_POLY_2 has found more than one **ensemble** directive in the CONTROL file.

Action:
Locate extra **ensemble** directives in CONTROL file and remove.

Message 416: error - conflicting force options in CONTROL file

DL_POLY_2 has found incompatible directives in the CONTROL file specifying the electrostatic interactions options.

Action:
Locate the conflicting directives in the CONTROL file and correct.

Message 418: error - bond vector work arrays too small in bndfrc

The work arrays in BNDFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 419: error - bond vector work arrays too small in angfrc

The work arrays in ANGFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 420: error - bond vector work arrays too small in tethfrc

The work arrays in TETHFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 421: error - bond vector work arrays too small in dihfrc

The work arrays in DIHFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 422: error - all-pairs must use multiple timestep

In DL_POLY_2 the 'all pairs' option must be used in conjunction with the multiple timestep.

Action:
Activate the multiple timestep option in the CONTROL file and resubmit.

Message 423: error - bond vector work arrays too small in shlfrc

The dimensions of the interatomic distance vectors have been exceeded in subroutine SHLFRC.

Action:
Standard user response. Fix the parameter `msbad`. Set equal to the value of the parameter `mxshl`.

Message 424: error - electrostatics incorrect for all-pairs

When using the all pairs option in conjunction with electrostatic forces, the electrostatics must be handled with either the standard Coulomb sum, or with the distance dependent dielectric.

Action:
Rerun the simulation with the appropriate electrostatic option.

Message 425: error - transfer buffer array too small in shlmerge

The buffer used to transfer data between nodes in the subroutine SHLMERGE has been dimensioned too small.

Action:
Standard user response. Fix the parameter `mxbuff`.

Message 426: error - neutral groups not permitted with all-pairs

DL_POLY_2 will not permit simulations using both the neutral group and all pairs options together.

Action:
Switch off one of the conflicting options and rerun.

Message 427: error - bond vector work arrays too small in invfrc

The work arrays in subroutine INVFRC have been exceeded.

Action:
Standard user response. Fix the parameter `msbad`.

Message 430: error - integration routine not available

A request for a nonexistent ensemble has been made or a request with conflicting options that DL_POLY_2 cannot deal with (e.g. a Evans thermostat with rigid body equations of motion).

Action:
Examine the CONTROL and FIELD files and remove inappropriate specifications.

Message 432: error - intlist failed to assign constraints

If the required simulation has constraint bonds DL_POLY_2 attempts to apportion the molecules to processors so that, if possible, there are no shared atoms between processors. If this is not possible, one or more molecules may be split between processors. This message

indicates that the code has failed to carry out either of these successfully.

Action:
The error may arise from a compiler error. Try recompiling INTLIST without the optimization flag turned on. If the problem persists it should be reported to the authors, (after checking the input data for inconsistencies).

Message 433: error - specify rcut before the Ewald sum precision

When specifying the desired precision for the Ewald sum in the CONTROL file, it is first necessary to specify the real space cutoff `rcut`.

Action:
Place the **cut** directive *before* the **ewald precision** directive in the CONTROL file and rerun.

Message 434: error - illegal entry into STRESS related routine

The calculation of the stress tensor in DL_POLY_2 requires additional code that must be included at compile time through the use of the STRESS keyword. If this is not done, and DL_POLY_2 is later required to calculate the stress tensor, this error will result.

Action:
The program must be recompiled with the STRESS keyword activated. This will ensure all the relevant code is in place. See section 3.2.1.

Message 436: error - unrecognised ensemble

An unknown ensemble option has been specified in the CONTROL file.

Action:
Locate **ensemble** directive in the CONTROL file and amend appropriately.

Message 438: error - PMF constraints failed to converge

The constraints in the potential of mean force algorithm have not converged in the permitted number of cycles. (The SHAKE algorithm for PMF constraints is iterative.) Possible causes include: a bad starting configuration; too large a time step used; incorrect force field specification; too high a temperature; inconsistent constraints involving shared atoms etc.

Action:
Corrective action depends on the cause. It is unlikely that simply increasing the iteration number will cure the problem, but you can try: follow standard user response to increase the parameter `mxshak`. But the trouble is much more likely to be cured by careful consideration

of the physical system being simulated. For example, is the system stressed in some way? Too far from equilibrium?

Message 440: error - undefined angular potential

A form of angular potential has been requested which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY_2 if this is possible. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and ANGFRC will be required.

Message 442: error - undefined three body potential

A form of three body potential has been requested which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY_2 if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and THBFRC will be required.

Message 443: error - undefined four body potential

DL_POLY_2 has been requested to process a four-body potential it does not recognise.

Action:
Check the FIELD file and make sure the keyword is correctly defined. Make sure that subroutine FBPFRC contains the code necessary to deal with the requested potential. Add the code required if necessary, by amending subroutines SYSDEF and FBPFRC.

Message 444: error - undefined bond potential

DL_POLY_2 has been requested to process a bond potential it does not recognise.

Action:
Check the FIELD file and make sure the keyword is correctly defined. Make sure that subroutine BNDFRC contains the code necessary to deal with the requested potential. Add the code required if necessary, by amending subroutines SYSDEF and BNDFRC.

Message 446: error - undefined electrostatic key in dihfrc

The subroutine DIHFRC has detected a request for an unknown kind of electrostatic model.

Action:
The probable source of the error is an improperly described force field. Check the CONTROL file and FIELD files for incompatible requirements.

Message 447: error - 1-4 separation exceeds cutoff range

In the subroutine DIHFRC the distance between the 1-4 atoms in the potential is larger than the cutoff that is applied to the 1-4 potential, meaning the potential will not be computed, though it may be an essential component of the dihedral force and not necessarily a vanishing force.

Action:
The probable source of the error is an improperly described force field. Effectively the 1-4 distance is not being restrained sufficently. Check the 1-4 potential parameters and the valence angles that help define the dihedral geometry. If these are correct, then you may have to comment out this error condition in the DIHFRC.F subroutine, but beware that when the 1-4 atoms are too widely separated, the dihedral angle can become indeterminable.

Message 448: error - undefined dihedral potential

A form of dihedral potential has been requested which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY_2 if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and DIHFRC (and its variants) will be required.

Message 449: error - undefined inversion potential

A form of inversion potential has been encountered which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY_2 if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and INVFRC will be required.

Message 450: error - undefined tethering potential

A form of tethering potential has been requested which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable

to DL_POLY_2 if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and TETHFRC will be required.

Message 451: error - three body potential cutoff undefined

The cutoff radius for a three body potential has not been defined in the FIELD file.

Action:
Locate the offending three body force potential in the FIELD file and add the required cutoff. Resubmit the job.

Message 452: error - undefined pair potential

A form of pair potential has been requested which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY_2 if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and FORGEN will be required.

Message 453: error - four body potential cutoff undefined

The cutoff radius for a four-body potential has not been defined in the FIELD file.

Action:
Locate the offending four body force potential in the FIELD file and add the required cutoff. Resubmit the job.

Message 454: error - undefined external field

A form of external field potential has been requested which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY_2 if this is reasonable. Alternatively, you may consider defining the required potential in the code yourself. Amendments to subroutines SYSDEF and EXTNFLD will be required.

Message 456: error - core and shell in same rigid unit

It is not sensible to fix both the core and the shell of a polarisable atom in the same molecular unit. Consequently DL_POLY_2 will abandon the job if this is found to be the case.

Action:
Locate the offending core-shell unit (there may be more than one in your FIELD file) and release the shell (preferably) from the rigid body specification.

Message 458: error - too many PMF constraints - param. mspmf too small

The number of constraints in the potential of mean force is too large. The dimensions of the appropriate arrays in DL_POLY_2 must be increased.

Action:
Standard user response. Fix the parameter `mspmf`.

Message 460: error - too many PMF sites - parameter mxspmf too small

The number of sites defined in the potential of mean force is too large. The dimensions of the appropriate arrays in DL_POLY_2 must be increased.

Action:
Standard user response. Fix the parameter `mxspmf`.

Message 461: error - undefined metal potential

The user has requested a metal potential DL_POLY_2 does not recognise.

Action:
Locate the metal potential specification in the FIELD file and replace with a recognised potential.

Message 462: error - PMF UNIT record expected

A **pmf unit** directive was expected as the next record in the FIELD file but was not found.

Action:
Locate the **pmf** directive in the FIELD file and examine the following entries. Insert the missing **pmf unit** directive and resubmit.

Message 463: error - unidentified atom in metal potential list

DL_POLY_2 checks all the metal potentials specified in the FIELD file and terminates the program if it can't identify any one of them from the atom types specified earlier in the file.

Action:
Correct the erroneous entry in the FIELD file and resubmit.

Message 465: error - calculated pair potential index too large

A zero or negative value for the thermostat time constant has been encountered in the CONTROL file.

Action:
Locate the **ensemble** directive in the CONTROL file and assign a positive value to the time constant.

Message 464: error - thermostat time constant must be > 0.d0

A zero or negative value for the thermostat time constant has been encountered in the CONTROL file.

Action:
Locate the **ensemble** directive in the CONTROL file and assign a positive value to the time constant.

Message 466: error - barostat time constant must be > 0.d0

A zero or negative value for the barostat time constant has been encountered in the CONTROL file.

Action:
Locate the **ensemble** directive in the CONTROL file and assign a positive value to the time constant.

Message 468: error - r0 too large for snm potential with current cutoff

The specified location (r0) of the potential minimum for a shifted n-m potential exceeds the specified potential cutoff. A potential with the desired minimum cannot be created.

Action:
To obtain a potential with the desired minimum it is necessary to increase the van der Waals cutoff. Locate the **rvdw** directive in the CONTROL file and reset to a magnitude greater than r0. Alternatively adjust the value of r0 in the FIELD file. Check that the FIELD file is correctly formatted.

Message 470: error - n<m in definition of n-m potential

The specification of a n-m potential in the FIELD file implies that the exponent m is larger than exponent n. (Not all versions of DL_POLY_2 are affected by this.)

Action:
Locate the n-m potential in the FIELD file and reverse the order of the exponents. Resubmit the job.

Message 474: error - mxxdf too small in parlst subroutine

The parameter `mxxdf` defining working arrays in subroutine PARLST of DL_POLY_2 has been found to be too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 475: error - mxxdf too small in parlst_nsq subroutine

The parameter `mxxdf` defining working arrays in subroutine PARLST_NSQ DL_POLY_2 has been found to be too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 476: error - mxxdf too small in parneulst subroutine

The parameter `mxxdf` defining working arrays in subroutine PARNEULST is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 477: error - mxxdf too small in prneulst subroutine

The parameter `mxxdf` defining working arrays in subroutine PRNEULST is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 478: error - mxxdf too small in forcesneu subroutine

The parameter `mxxdf` defining working arrays in subroutine FORCESNEU is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 479: error - mxxdf too small in multipleneu subroutine

The parameter `mxxdf` defining working arrays in subroutine MULTIPLENEU is too small.

Action:
Standard user response. Fix the parameter `mxxdf`.

Message 484: error - only one potential of mean force permitted

It is not permitted to define more than one potential of mean force in the FIELD file.

Action:
Check that the FIELD file contains only one PMF specification. If more than one is needed, DL_POLY_2 cannot handle it.

Message 486: error - HK real space screening function cutoff violation

DL_POLY_2 has detected an unacceptable degree of inaccuracy in the screening function near the radius of cutoff *in real space*, which implies the Hautman-Klein Ewald method will not be sufficiently accurate.

Action:
The user should respecify the HK control parameters given in the CONTROL file. Either the convergence parameter should be increased or the sum expanded to incorporate more images of the central cell. (Warning: increasing the convergence parameter may cause failure in the reciprocal space domain.) (See 4.1.1).

Message 487: error - HK recip space screening function cutoff violation

DL_POLY_2 has detected an unacceptable degree of inaccuracy in the screening function near the radius of cutoff *in reciprocal space*, which implies the Hautman-Klein Ewald method will not be sufficiently accurate.

Action:
The user should respecify the HK control parameters given in the CONTROL file. Either the convergence parameter should be reduced or more k-vectors used. (Warning: reducing the convergence parameter may cause failure in the real space domain.) (See 4.1.1).

Message 488: error - HK lattice control parameter set too large

The Hautman-Klein Ewald method in DL_POLY_2 permits the user to perform a real space sum over nearest-neighbour and next-nearest-neighbour cells (i.e. up to `nlatt=2`). If the user specifies a larger sum than this, this error will result.

Action:
The user should respecify the HK control parameters given in the CONTROL file and set `nlatt` to a maximum of 2. (See 4.1.1).

Message 490: error - PMF parameter mxpmf too small in passpmf

The bookkeeping arrays have been exceeded in PASSPMF

Action:
Standard user response. Fix the parameter `mxpmf`. Set equal to `mxatms`.

Message 492: error - parameter mxcons < number of PMF constraints

The parameter `mxcons` is too small for the number of PMF constraints in the system.

Action:
Standard user response. Fix the value of `mxcons`.

Message 494: error in csend: pvmfinitsend

The PVM routine PVMFINITSEND has returned an error. It is invoked by the routine CSEND.

Action:
Check your system implementation of PVM.

Message 496: error in csend: pvmfpack

The PVM routine PVMFPACK has returned an error. It is invoked by the routine CSEND.

Action:
Check your system implementation of PVM.

Message 498: error in csend: pvmfsend

The PVM routine PVMFSEND has returned an error. It is invoked by the routine CSEND.

Action:
Check your system implementation of PVM.

Message 500: error in crecv: pvmfrecv

The PVM routine PVMFRECV has returned an error. It is invoked by the routine CRECV.

Action:
Check your system implementation of PVM.

Message 502: error in crecv: pvmfunpack

The PVM routine PVMFUNPACK has returned an error. It is invoked by the routine CRECV.

Action:
Check your system implementation of PVM.

Message 504: error - cutoff too large for TABLE file

The requested cutoff exceeds the information in the TABLE file.

Action:
Reduce the value of the vdw cutoff (`rvdw`) in the CONTROL file or reconstruct the TABLE file.

Message 506: error - work arrays too small for quaternion integration

The working arrays associated with quaternions are too small for the size of system being simulated. They must be redimensioned.

Action:
Standard user response. Fix the parameter `msgrp`.

Message 508: error - rigid bodies not permitted with RESPA algorithm

The RESPA algorithm implemented in DL_POLY_2 is for atomic systems only. Rigid bodies or constraints cannot be treated.

Action:
There is no cure for this. The code simply does not have this capability. Consider writing it for yourself!

Message 510: error - structure optimiser not permitted with RESPA

The RESPA algorithm in DL_POLY_2 has not been implemented to work with the structure optimizer. You have asked for a forbidden operation.

Action:
There is no fix for this. In any case it does not make sense to use the RESPA algorithm for this purpose.

Message 513: error - SPME not available for given boundary conditions

The SPME algorithm in DL_POLY_2 does not work for aperiodic (IMCON=0) or slab (IMCON=6) boundary conditions.
Action:
If the system must have aperiodic or slab boundaries, nothing can be done. In the latter case however, it may be acceptable to represent the same system with slabs replicated in the z direction, thus permitting a periodic simulation.

Message 514: error - SPME routines have not been compiled in

The inclusion of the SPME algorithm in DL_POLY_2 is optional at the compile stage. If the executable does not contain the SPME routines, but the method is requested by the user, this error results.

Action:
DL_POLY_2 must be recompiled with the SPME flags set. Beware that your system has the necessary fast Fourier transform routines to permit this.

Message 1000: error - failed allocation of configuration arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1010: error - failed allocation of angle arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1011: error - failed allocation of dihedral arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1012: error - failed allocation of exclude arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1013: error - failed allocation of rigid body arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1014: error - failed allocation of vdw arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1020: error - failed allocation of angle work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1030: error - failed allocation of bond arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1040: error - failed allocation of bond work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1050: error - failed allocation of dihedral arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1060: error - failed allocation of dihedral work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1070: error - failed allocation of constraint arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1090: error - failed allocation of site arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1100: error - failed allocation of core_shell arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1110: error - failed allocation of core_shell work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1120: error - failed allocation of inversion arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1130: error - failed allocation of inversion work arrays'

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1140: error - failed allocation of four-body arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1150: error - failed allocation of four-body work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1170: error - failed allocation of three-body arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1180: error - failed allocation of three-body work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1200: error - failed allocation of external field arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1210: error - failed allocation of pmf arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1220: error - failed allocation of pmf_lf or pmf_vv work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1230: error - failed allocation of pmf_shake work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1240: error - failed allocation of ewald arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1250: error - failed allocation of excluded atom arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1260: error - failed allocation of tethering arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1270: error - failed allocation of tethering work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1280: error - failed allocation of metal arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1290: error - failed allocation of work arrays in nvt_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1300: error - failed allocation of dens0 array in npt_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1310: error - failed allocation of work arrays in npt_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1320: error - failed allocation of dens0 array in npt_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1330: error - failed allocation of work arrays in npt_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1340: error - failed allocation of dens0 array in nst_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1350: error - failed allocation of work arrays in nst_b0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1360: error - failed allocation of dens0 array in nst_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1370: error - failed allocation of work arrays in nst_h0.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1380: error - failed allocation of work arrays in nve_1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1390: error - failed allocation of work arrays in nvt_e1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1400: error - failed allocation of work arrays in nvt_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1410: error - failed allocation of work arrays in nvt_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1420: error - failed allocation of work arrays in npt_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1430: error - failed allocation of density array in npt_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1440: error - failed allocation of work arrays in npt_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1450: error - failed allocation of density array in npt_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1460: error - failed allocation of work arrays in nst_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1470: error - failed allocation of density array in nst_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1480: error - failed allocation of work arrays in nst_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1490: error - failed allocation of density array in nst_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1500: error - failed allocation of work arrays in nveq_1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1510: error - failed allocation of work arrays in nvtq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1520: error - failed allocation of work arrays in nvtq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1530: error - failed allocation of work arrays in nptq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1540: error - failed allocation of density array in nptq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1550: error - failed allocation of work arrays in nptq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1560: error - failed allocation of density array in nptq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1570: error - failed allocation of work arrays in nstq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1580: error - failed allocation of density array in nstq_b1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1590: error - failed allocation of work arrays in nstq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1600: error - failed allocation of density array in nstq_h1.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1610: error - failed allocation of work arrays in qshake.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1615: error - failed allocation of work arrays in qrattle_q.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1620: error - failed allocation of work arrays in nveq_2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1625: error - failed allocation of work arrays in qrattle_v.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1630: error - failed allocation of work arrays in nvtq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1640: error - failed allocation of work arrays in nvtq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1650: error - failed allocation of work arrays in nptq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1660: error - failed allocation of density array in nptq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1670: error - failed allocation of work arrays in nptq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1680: error - failed allocation of density array in nptq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1690: error - failed allocation of work arrays in nstq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1700: error - failed allocation of density array in nstq_b2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1710: error - failed allocation of work arrays in nstq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1720: error - failed allocation of density array in nstq_h2.f

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1730: error - failed allocation of HK Ewald arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1740: error - failed allocation of property arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1750: error - failed allocation of spme arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1760: error - failed allocation of ewald_spme.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1770: error - failed allocation of quench.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1780: error - failed allocation of quatqnch.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1790: error - failed allocation of quatbook.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1800: error - failed allocation of intlist.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1810: error - failed allocation of forces.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1820: error - failed allocation of forcesneu.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1830: error - failed allocation of neutlst.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1840: error - failed allocation of multiple.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1850: error - failed allocation of multipleneu.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1860: error - failed allocation of multiple_nsq.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1870: error - failed allocation of parlst_nsq.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1880: error - failed allocation of parlst.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1890: error - failed allocation of parlink.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1900: error - failed allocation of parlinkneu.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1910: error - failed allocation of parneulst.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1920: error - failed allocation of strucopt.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1930: error - failed allocation of vertest.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1940: error - failed allocation of pair arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1945: error - failed allocation of tersoff arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1950: error - shell relaxation cycle limit exceeded

There has been a convergence failure during the execution of relaxed shell polarisation model. Probable cause: the system is unstable e.g. in an abnormally high energy configuration.
Action:
Increasing the maximum number of cycles permitted in the shell relaxation set by variable `mxpass` in the dlpoly.f root program may help, but it is unlikely. A better option is to relax the structure somehow first e.g. using the **zero** option in the CONTROL file.

Message 1953: error - tersoff radius of cutoff not defined

The Tersoff potential requires the user to specify a short ranged cutoff as part of the potential description. This is distinct from the normal cutoff used by the Van der Waals interactions.
Action:
Check the Tersoff potential description in the FIELD file. Make sure it is fully complete.

Message 1955: error - failed allocation of tersoff work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1960: error - conflicting shell option in FIELD file

The relaxed shell and adiabatic shell polarisation options in DL_POLY_2 are mutually exclusive. The user has request both options in the same simulation.
Action:
Locate the occurrences of the **shell** directives in the FIELD file and ensure they specify the same shell model.

Message 1970: error - failed allocation of shell_relax work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1972: error - unknown tersoff potential defined in FIELD file

DL_POLY_2 has failed to recognise the Tersoff potentials specified by the user in the FIELD file.

Action:
Locate the Tersoff potential specification in the FIELD fiel and ensure it is correctly defined.

Message 1974: error - incorrect period boundary in tersoff.f

The implementation of the Tersoff potential in DL_POLY_2 is based on the link cell algorithm, which is suitable for rectangular or triclinic MD cells only. It is not suitable for any other shape of MD cell.
Action:
The user must reconstruct the system according to one of the permitted periodic boundaries.

Message 1976: error - too many link cells required in tersoff.f

The number of link cells required by the Tersoff routines exceeds the amount allowed for by DL_POLY_2 . This can happen if the system is simulated under NPT or NST conditions and the system volume increases dramatically.
Action:
The problem may cure itself on restart, provided the restart configuration has already expande significantly. Otherwise the user must locate and adjust the `mxcell` according to the standard response procedure.

Message 1978: error - undefined potential in tersoff.f

A form of Tersoff potential has been requested which DL_POLY_2 does not recognise.

Action:
Locate the offending potential in the FIELD file and remove. Replace with one acceptable to DL_POLY_2 if this is reasonable. Alternatively, you may consider defining the required

potential in the code yourself. Amendments to subroutines SYSDEF and TERSOFF will be required.

Message 1980: error - failed allocation of nvevv_1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 1990: error - failed allocation of nvtvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2000: error - failed allocation of nvtvv_e1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2010: error - failed allocation of nvtvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2020: error - failed allocation of nptvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2030: error - failed allocation of nptvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2040: error - failed allocation of nptvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2050: error - failed allocation of nptvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2060: error - failed allocation of nstvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2070: error - failed allocation of nstvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2080: error - failed allocation of nstvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2090: error - failed allocation of nstvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2100: error - failed allocation of nveqvv_1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2110: error - failed allocation of nveqvv_2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2120: error - failed allocation of nvtqvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2130: error - failed allocation of nvtqvv_b2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2140: error - failed allocation of nvtqvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2150: error - failed allocation of nvtqvv_h2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2160: error - failed allocation of nptqvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2170: error - failed allocation of nptqvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2180: error - failed allocation of nptqvv_b2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2190: error - failed allocation of nptqvv_b2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2200: error - failed allocation of nptqvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2210: error - failed allocation of nptqvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2220: error - failed allocation of nptqvv_h2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2230: error - failed allocation of nptqvv_h2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2240: error - failed allocation of nstqvv_b1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2250: error - failed allocation of nstqvv_b1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2260: error - failed allocation of nstqvv_b2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2270: error - failed allocation of nstqvv_b2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2280: error - failed allocation of nstqvv_h1.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2290: error - failed allocation of nstqvv_h1.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2300: error - failed allocation of nstqvv_h2.f dens0 array

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Message 2310: error - failed allocation of nstqvv_h2.f work arrays

This is a memory allocation error. Probable cause: excessive size of simulated system.

Action:
If the simulated system cannot be replaced by a smaller one, the user must consider using more processors or a machine with larger memory per processor.

Appendix D

Subroutine Locations

The Locations of Subroutines and Functions

The following table lists the subroutines and functions in DL_POLY_2 and which source files they can be found in.

Routine	Kind	Location
abort_config_read	subroutine	define_system.f
abort_control_read	subroutine	define_system.f
abort_eamtable_read	subroutine	metal_terms.f
abort_field_read	subroutine	define_system.f
abort_table_read	subroutine	vdw_terms.f
abortscan	subroutine	setup_program.f
alloc_ang_arrays	subroutine	angles_module.f
alloc_bnd_arrays	subroutine	bonds_module.f
alloc_config_arrays	subroutine	config_module.f
alloc_csh_arrays	subroutine	core_shell_module.f
alloc_dih_arrays	subroutine	dihed_module.f
alloc_ewald_arrays	subroutine	ewald_module.f
alloc_exc_arrays	subroutine	exclude_module.f
alloc_fbp_arrays	subroutine	four_body_module.f
alloc_fld_arrays	subroutine	external_field_module.f
alloc_hke_arrays	subroutine	hkewald_module.f
alloc_inv_arrays	subroutine	inversion_module.f
alloc_met_arrays	subroutine	metal_module.f
alloc_pair_arrays	subroutine	pair_module.f
alloc_pmf_arrays	subroutine	pmf_module.f
alloc_prp_arrays	subroutine	property_module.f
alloc_rgbdy_arrays	subroutine	rigid_body_module.f
alloc_shake_arrays	subroutine	shake_module.f
alloc_site_arrays	subroutine	site_module.f
alloc_spme_arrays	subroutine	spme_module.f

alloc_tbp_arrays	subroutine	three_body_module.f
alloc_ter_arrays	subroutine	tersoff_module.f
alloc_tet_arrays	subroutine	tether_module.f
alloc_vdw_arrays	subroutine	vdw_module.f
angfrc	subroutine	angle_terms.f
bndfrc	subroutine	bond_terms.f
bodystress	subroutine	rigid_body_terms.f
bspcoe	subroutine	spme_terms.f
bspgen	subroutine	spme_terms.f
cell_propagate	subroutine	ensemble_tools.f
cell_update	subroutine	ensemble_tools.f
cerfr	function	hkewald_terms.f
cfgscan	subroutine	setup_program.f
check_shells	subroutine	core_shell_terms.f
check_syschg	subroutine	site_terms.f
conscan	subroutine	setup_program.f
copystring	subroutine	parse_tools.f
corshl	subroutine	core_shell_terms.f
coul0	subroutine	coulomb_terms.f
coul0neu	subroutine	neu_coul_terms.f
coul1	subroutine	coulomb_terms.f
coul2	subroutine	coulomb_terms.f
coul2neu	subroutine	neu_coul_terms.f
coul3	subroutine	coulomb_terms.f
coul3neu	subroutine	neu_coul_terms.f
coul4	subroutine	coulomb_terms.f
cpy_rtc	subroutine	utility_pack.f
crecv	subroutine	basic_comms.f
crecv	subroutine	serial.f
csend	subroutine	basic_comms.f
csend	subroutine	serial.f
dblstr	function	parse_tools.f
dcell	subroutine	utility_pack.f
define_angles	subroutine	angle_terms.f
define_atoms	subroutine	site_terms.f
define_bonds	subroutine	bond_terms.f
define_constraints	subroutine	shake_terms.f
define_core_shell	subroutine	core_shell_terms.f
define_dihedrals	subroutine	dihedral_terms.f
define_external_field	subroutine	external_field_terms.f
define_four_body	subroutine	four_body_terms.f
define_inversions	subroutine	inversion_terms.f
define_metals	subroutine	metal_terms.f
define_pmf	subroutine	pmf_terms.f

define_rigid_body	subroutine	rigid_body_terms.f
define_tersoff	subroutine	tersoff_terms.f
define_tethers	subroutine	tether_terms.f
define_three_body	subroutine	three_body_terms.f
define_units	subroutine	define_system.f
define_van_der_waals	subroutine	vdw_terms.f
diffsn0	subroutine	system_properties.f
diffsn1	subroutine	system_properties.f
dihfrc	subroutine	dihedral_terms.f
dlpfft3	subroutine	spme_terms.f
duni	function	utility_pack.f
eamden	subroutine	metal_terms.f
ele_prd	subroutine	utility_pack.f
erfcgen	subroutine	ewald_terms.f
error	subroutine	error.f
ewald1	subroutine	ewald_terms.f
ewald2	subroutine	ewald_terms.f
ewald2neu	subroutine	neu_ewald_terms.f
ewald3	subroutine	ewald_terms.f
ewald3_neu	subroutine	neu_ewald_terms.f
ewald4	subroutine	ewald_terms.f
ewald_spme	subroutine	spme_terms.f
exclude	subroutine	exclude_terms.f
exclude_atom	subroutine	exclude_terms.f
exclude_link	subroutine	exclude_terms.f
excludeneu	subroutine	exclude_terms.f
exitcomms	subroutine	basic_comms.f
exitcomms	subroutine	serial.f
extnfld	subroutine	external_field_terms.f
fbpfrc	subroutine	four_body_terms.f
fcap	subroutine	utility_pack.f
findstring	function	parse_tools.f
fldscan	subroutine	setup_program.f
forces	subroutine	force_drivers.f
forcesneu	subroutine	force_drivers.f
forgen	subroutine	vdw_terms.f
fortab	subroutine	vdw_terms.f
freeze	subroutine	utility_pack.f
fsden	subroutine	metal_terms.f
gauss	subroutine	utility_pack.f
gdsum	subroutine	basic_comms.f
gdsum	subroutine	serial.f
getcom	subroutine	ensemble_tools.f
getkin	function	kinetic_terms.f

getkinf	function	kinetic_terms.f
getking	subroutine	kinetic_terms.f
getkinr	function	kinetic_terms.f
getkins	subroutine	kinetic_terms.f
getkint	function	kinetic_terms.f
getmass	function	ensemble_tools.f
getrec	subroutine	parse_tools.f
getrotmat	subroutine	utility_pack.f
getvom	subroutine	ensemble_tools.f
getword	subroutine	parse_tools.f
gimax	subroutine	basic_comms.f
gimax	subroutine	serial.f
gisum	subroutine	basic_comms.f
gisum	subroutine	serial.f
global_sum_forces	subroutine	utility_pack.f
gstate	subroutine	basic_comms.f
gstate	subroutine	serial.f
gsync	subroutine	basic_comms.f
gsync	subroutine	serial.f
hkewald1	subroutine	hkewald_terms.f
hkewald2	subroutine	hkewald_terms.f
hkewald3	subroutine	hkewald_terms.f
hkewald4	subroutine	hkewald_terms.f
hkgen	subroutine	hkewald_terms.f
images	subroutine	utility_pack.f
initcomms	subroutine	basic_comms.f
initcomms	subroutine	serial.f
intlist	subroutine	define_system.f
intstr	function	parse_tools.f
intstr3	function	utility_pack.f
invert	subroutine	utility_pack.f
invfrc	subroutine	inversion_terms.f
jacobi	subroutine	utility_pack.f
kinstr	subroutine	utility_pack.f
kinstress	subroutine	kinetic_terms.f
kinstressf	subroutine	kinetic_terms.f
kinstressg	subroutine	kinetic_terms.f
lf_integrate	subroutine	lf_integrate.f
loc8	function	utility_pack.f
lowcase	subroutine	parse_tools.f
lrcmetal	subroutine	metal_terms.f
lrcorrect	subroutine	vdw_terms.f
machine	subroutine	basic_comms.f
machine	subroutine	serial.f

matmul	subroutine	utility_pack.f
merge	subroutine	merge_hcube.f
merge	subroutine	merge_systol.f
merge	subroutine	merge_tools.f
merge	subroutine	serial.f
merge1	subroutine	merge_hcube.f
merge1	subroutine	merge_systol.f
merge1	subroutine	merge_tools.f
merge1	subroutine	serial.f
merge4	subroutine	merge_hcube.f
merge4	subroutine	merge_systol.f
merge4	subroutine	merge_tools.f
merge4	subroutine	serial.f
metal_deriv	subroutine	metal_terms.f
metdens	subroutine	metal_terms.f
metfrc	subroutine	metal_terms.f
metgen	subroutine	metal_terms.f
mettab	subroutine	metal_terms.f
mkwd8	function	parse_tools.f
multiple	subroutine	force_drivers.f
multiple_nsq	subroutine	force_drivers.f
multipleneu	subroutine	force_drivers.f
mynode	function	basic_comms.f
mynode	function	serial.f
neutbook	subroutine	define_system.f
neutlst	subroutine	force_drivers.f
nodedim	function	basic_comms.f
nodedim	function	serial.f
nosquish	subroutine	vv_rotation_1.f
npt_b1	subroutine	lf_motion_1.f
npt_h1	subroutine	lf_motion_1.f
nptq_b1	subroutine	lf_rotation_1.f
nptq_b2	subroutine	lf_rotation_2.f
nptq_h1	subroutine	lf_rotation_1.f
nptq_h2	subroutine	lf_rotation_2.f
nptqscl_p	subroutine	ensemble_tools.f
nptqscl_t	subroutine	ensemble_tools.f
nptqvv_b1	subroutine	vv_rotation_1.f
nptqvv_b2	subroutine	vv_rotation_2.f
nptqvv_h1	subroutine	vv_rotation_1.f
nptqvv_h2	subroutine	vv_rotation_2.f
nptscale_p	subroutine	ensemble_tools.f
nptscale_t	subroutine	ensemble_tools.f
nptvv_b1	subroutine	vv_motion_1.f

```
nptvv_h1              subroutine   vv_motion_1.f
nst_b1                subroutine   lf_motion_1.f
nst_h1                subroutine   lf_motion_1.f
nstq_b1               subroutine   lf_rotation_1.f
nstq_b2               subroutine   lf_rotation_2.f
nstq_h1               subroutine   lf_rotation_1.f
nstq_h2               subroutine   lf_rotation_2.f
nstqmtk_p             subroutine   ensemble_tools.f
nstqscl_p             subroutine   ensemble_tools.f
nstqscl_p2            subroutine   ensemble_tools.f
nstqscl_t             subroutine   ensemble_tools.f
nstqscl_t2            subroutine   ensemble_tools.f
nstqvv_b1             subroutine   vv_rotation_1.f
nstqvv_b2             subroutine   vv_rotation_2.f
nstqvv_h1             subroutine   vv_rotation_1.f
nstqvv_h2             subroutine   vv_rotation_2.f
nstscale_p            subroutine   ensemble_tools.f
nstscale_t            subroutine   ensemble_tools.f
nstvv_b1              subroutine   vv_motion_1.f
nstvv_h1              subroutine   vv_motion_1.f
numnodes              function     basic_comms.f
numnodes              function     serial.f
nve_1                 subroutine   lf_motion_1.f
nveq_1                subroutine   lf_rotation_1.f
nveq_2                subroutine   lf_rotation_2.f
nveqvv_1              subroutine   vv_rotation_1.f
nveqvv_2              subroutine   vv_rotation_2.f
nvevv_1               subroutine   vv_motion_1.f
nvt_b1                subroutine   lf_motion_1.f
nvt_e1                subroutine   lf_motion_1.f
nvt_h1                subroutine   lf_motion_1.f
nvtq_b1               subroutine   lf_rotation_1.f
nvtq_b2               subroutine   lf_rotation_2.f
nvtq_h1               subroutine   lf_rotation_1.f
nvtq_h2               subroutine   lf_rotation_2.f
nvtqscl               subroutine   ensemble_tools.f
nvtqvv_b1             subroutine   vv_rotation_1.f
nvtqvv_b2             subroutine   vv_rotation_2.f
nvtqvv_h1             subroutine   vv_rotation_1.f
nvtqvv_h2             subroutine   vv_rotation_2.f
nvtscale              subroutine   ensemble_tools.f
nvtvv_b1              subroutine   vv_motion_1.f
nvtvv_e1              subroutine   vv_motion_1.f
nvtvv_h1              subroutine   vv_motion_1.f
```

parlink	subroutine	nlist_builders.f
parlinkneu	subroutine	nlist_builders.f
parlst	subroutine	nlist_builders.f
parlst_nsq	subroutine	nlist_builders.f
parneulst	subroutine	nlist_builders.f
parset	subroutine	setup_program.f
passcon	subroutine	pass_tools.f
passcon	subroutine	serial.f
passpmf	subroutine	pass_tools.f
passpmf	subroutine	serial.f
passquat	subroutine	pass_tools.f
passquat	subroutine	serial.f
pivot	subroutine	vv_rotation_2.f
pmf_rattle_r	subroutine	pmf_vv.f
pmf_rattle_v	subroutine	pmf_vv.f
pmf_shake	subroutine	pmf_lf.f
pmf_vectors	subroutine	pmf_terms.f
pmflf	subroutine	pmf_lf.f
pmfvv	subroutine	pmf_vv.f
primlst	subroutine	nlist_builders.f
prneulst	subroutine	nlist_builders.f
put_shells_on_cores	subroutine	core_shell_terms.f
qrattle_r	subroutine	vv_rotation_2.f
qrattle_v	subroutine	vv_rotation_2.f
qshake	subroutine	lf_rotation_2.f
quatbook	subroutine	define_system.f
quatqnch	subroutine	temp_scalers.f
quench	subroutine	temp_scalers.f
rdf0	subroutine	system_properties.f
rdf0neu	subroutine	system_properties.f
rdf1	subroutine	system_properties.f
rdrattle_r	subroutine	vv_motion_1.f
rdrattle_v	subroutine	vv_motion_1.f
rdshake_1	subroutine	lf_motion_1.f
relax_shells	subroutine	core_shell_terms.f
result	subroutine	system_properties.f
revive	subroutine	system_properties.f
rotate_omega	subroutine	vv_rotation_1.f
scl_csum	subroutine	utility_pack.f
sdot0	function	utility_pack.f
sdot1	function	utility_pack.f
set_block	subroutine	utility_pack.f
shellsort	subroutine	utility_pack.f
shlfrc	subroutine	core_shell_terms.f

shlmerge	subroutine	merge_hcube.f
shlmerge	subroutine	merge_systol.f
shlmerge	subroutine	merge_tools.f
shlmerge	subroutine	serial.f
shlqnch	subroutine	temp_scalers.f
shmove	subroutine	merge_hcube.f
shmove	subroutine	merge_systol.f
shmove	subroutine	merge_tools.f
shmove	subroutine	serial.f
simdef	subroutine	define_system.f
spl_cexp	subroutine	spme_terms.f
splice	subroutine	merge_hcube.f
splice	subroutine	merge_systol.f
splice	subroutine	merge_tools.f
splice	subroutine	serial.f
spme_for	subroutine	spme_terms.f
srfrce	subroutine	vdw_terms.f
srfrceneu	subroutine	vdw_terms.f
static	subroutine	system_properties.f
strip	subroutine	parse_tools.f
striptext	subroutine	parse_tools.f
strucopt	subroutine	strucopt.f
sysbook	subroutine	define_system.f
sysdef	subroutine	define_system.f
sysgen	subroutine	define_system.f
sysinit	subroutine	define_system.f
systemp	subroutine	define_system.f
tergen	subroutine	tersoff_terms.f
terint	subroutine	tersoff_terms.f
tersoff	subroutine	tersoff_terms.f
tersoff3	subroutine	tersoff_terms.f
tethfrc	subroutine	tether_terms.f
thbfrc	subroutine	three_body_terms.f
timchk	subroutine	timchk.f
traject	subroutine	traject.f
traject	subroutine	traject_u.f
update_quaternions	subroutine	lf_rotation_1.f
vertest	subroutine	nlist_builders.f
vscaleg	subroutine	temp_scalers.f
vv_integrate	subroutine	vv_integrate.f
warning	subroutine	warning.f
xscale	subroutine	utility_pack.f
zden0	subroutine	system_properties.f
zden1	subroutine	system_properties.f

Appendix E

Called Subroutines

Subroutines Called

The following table lists the subroutines in DL_POLY_2 and which subroutines they call.

Calling routine	Called routine
angle_terms.f	copystring
angle_terms.f	dblstr
angle_terms.f	error
angle_terms.f	gdsum
angle_terms.f	getrec
angle_terms.f	getword
angle_terms.f	gstate
angle_terms.f	images
angle_terms.f	intstr
angle_terms.f	lowcase
angles_module.f	error
basic_comms.f	MPI_BARRIER
basic_comms.f	MPI_COMM_RANK
basic_comms.f	MPI_COMM_SIZE
basic_comms.f	MPI_FINALIZE
basic_comms.f	MPI_RECV
basic_comms.f	MPI_allreduce
basic_comms.f	MPI_init
basic_comms.f	MPI_send
basic_comms.f	exit
basic_comms.f	gisum
basic_comms.f	mynode
basic_comms.f	numnodes
bond_terms.f	copystring
bond_terms.f	dblstr
bond_terms.f	error

```
bond_terms.f            | gdsum
bond_terms.f            | getrec
bond_terms.f            | getword
bond_terms.f            | gstate
bond_terms.f            | images
bond_terms.f            | intstr
bond_terms.f            | lowcase
bonds_module.f          | error
config_module.f         | error
core_shell_module.f     | error
core_shell_terms.f      | dblstr
core_shell_terms.f      | error
core_shell_terms.f      | gdsum
core_shell_terms.f      | getrec
core_shell_terms.f      | images
core_shell_terms.f      | intstr
core_shell_terms.f      | merge
core_shell_terms.f      | merge1
define_system.f         | abort_config_read
define_system.f         | abort_control_read
define_system.f         | abort_field_read
define_system.f         | ccfft3d
define_system.f         | check_shells
define_system.f         | check_syschg
define_system.f         | copystring
define_system.f         | dblstr
define_system.f         | dcell
define_system.f         | define_angles
define_system.f         | define_atoms
define_system.f         | define_bonds
define_system.f         | define_constraints
define_system.f         | define_core_shell
define_system.f         | define_dihedrals
define_system.f         | define_external_field
define_system.f         | define_four_body
define_system.f         | define_inversions
define_system.f         | define_metals
define_system.f         | define_pmf
define_system.f         | define_rigid_body
define_system.f         | define_tersoff
define_system.f         | define_tethers
define_system.f         | define_three_body
define_system.f         | define_units
define_system.f         | define_van_der_waals
```

```
define_system.f          | error
define_system.f          | exclude
define_system.f          | exclude_atom
define_system.f          | exclude_link
define_system.f          | excludeneu
define_system.f          | fftw3d_f77_create_plan
define_system.f          | findstring
define_system.f          | gauss
define_system.f          | gdsum
define_system.f          | getrec
define_system.f          | gimax
define_system.f          | gstate
define_system.f          | images
define_system.f          | intlist
define_system.f          | intstr
define_system.f          | invert
define_system.f          | jacobi
define_system.f          | lowcase
define_system.f          | lrcmetal
define_system.f          | lrcorrect
define_system.f          | merge
define_system.f          | merge4
define_system.f          | mkwd8
define_system.f          | neutbook
define_system.f          | passcon
define_system.f          | passpmf
define_system.f          | passquat
define_system.f          | put_shells_on_cores
define_system.f          | quatbook
define_system.f          | quatqnch
define_system.f          | quench
define_system.f          | shellsort
define_system.f          | shlqnch
define_system.f          | strip
define_system.f          | vscaleg
define_system.f          | warning
define_system.f          | zzfft3d
dihed_module.f           | error
dihedral_terms.f         | copystring
dihedral_terms.f         | dblstr
dihedral_terms.f         | error
dihedral_terms.f         | gdsum
dihedral_terms.f         | getrec
dihedral_terms.f         | getword
```

```
dihedral_terms.f        | gstate
dihedral_terms.f        | images
dihedral_terms.f        | intstr
dihedral_terms.f        | lowcase
dlpoly.f                | alloc_ang_arrays
dlpoly.f                | alloc_bnd_arrays
dlpoly.f                | alloc_config_arrays
dlpoly.f                | alloc_csh_arrays
dlpoly.f                | alloc_dih_arrays
dlpoly.f                | alloc_ewald_arrays
dlpoly.f                | alloc_exc_arrays
dlpoly.f                | alloc_fbp_arrays
dlpoly.f                | alloc_fld_arrays
dlpoly.f                | alloc_hke_arrays
dlpoly.f                | alloc_inv_arrays
dlpoly.f                | alloc_met_arrays
dlpoly.f                | alloc_pair_arrays
dlpoly.f                | alloc_pmf_arrays
dlpoly.f                | alloc_prp_arrays
dlpoly.f                | alloc_rgbdy_arrays
dlpoly.f                | alloc_shake_arrays
dlpoly.f                | alloc_site_arrays
dlpoly.f                | alloc_spme_arrays
dlpoly.f                | alloc_tbp_arrays
dlpoly.f                | alloc_ter_arrays
dlpoly.f                | alloc_tet_arrays
dlpoly.f                | alloc_vdw_arrays
dlpoly.f                | angfrc
dlpoly.f                | bndfrc
dlpoly.f                | corshl
dlpoly.f                | dcell
dlpoly.f                | dihfrc
dlpoly.f                | error
dlpoly.f                | exitcomms
dlpoly.f                | extnfld
dlpoly.f                | fbpfrc
dlpoly.f                | fcap
dlpoly.f                | forces
dlpoly.f                | forcesneu
dlpoly.f                | freeze
dlpoly.f                | gdsum
dlpoly.f                | global_sum_forces
dlpoly.f                | gsync
dlpoly.f                | initcomms
```

```
dlpoly.f                | invfrc
dlpoly.f                | kinstress
dlpoly.f                | lf_integrate
dlpoly.f                | lrcmetal
dlpoly.f                | machine
dlpoly.f                | multiple
dlpoly.f                | multiple_nsq
dlpoly.f                | multipleneu
dlpoly.f                | parlink
dlpoly.f                | parlinkneu
dlpoly.f                | parlst
dlpoly.f                | parlst_nsq
dlpoly.f                | parneulst
dlpoly.f                | parset
dlpoly.f                | quatqnch
dlpoly.f                | relax_shells
dlpoly.f                | result
dlpoly.f                | revive
dlpoly.f                | shlfrc
dlpoly.f                | shlqnch
dlpoly.f                | simdef
dlpoly.f                | static
dlpoly.f                | strucopt
dlpoly.f                | sysbook
dlpoly.f                | sysdef
dlpoly.f                | sysgen
dlpoly.f                | sysinit
dlpoly.f                | systemp
dlpoly.f                | tersoff
dlpoly.f                | tethfrc
dlpoly.f                | thbfrc
dlpoly.f                | timchk
dlpoly.f                | traject
dlpoly.f                | vertest
dlpoly.f                | vscaleg
dlpoly.f                | vv_integrate
dlpoly.f                | xscale
dlpoly.f                | zden0
ensemble_tools.f        | cell_update
ensemble_tools.f        | dcell
ensemble_tools.f        | gdsum
ensemble_tools.f        | getkin
ensemble_tools.f        | getkinf
ensemble_tools.f        | getking
```

```
ensemble_tools.f         | getking
ensemble_tools.f         | invert
ensemble_tools.f         | kinstress
ensemble_tools.f         | kinstressf
ensemble_tools.f         | kinstressg
ensemble_tools.f         | matmul
ensemble_tools.f         | sdot0
error.f                  | exitcomms
error.f                  | gsync
ewald_module.f           | error
ewald_terms.f            | dcell
ewald_terms.f            | error
ewald_terms.f            | gdsum
ewald_terms.f            | invert
exclude_module.f         | error
exclude_terms.f          | error
exclude_terms.f          | gimax
exclude_terms.f          | gstate
exclude_terms.f          | warning
external_field_module.f  | error
external_field_terms.f   | copystring
external_field_terms.f   | dblstr
external_field_terms.f   | error
external_field_terms.f   | getrec
external_field_terms.f   | getword
external_field_terms.f   | intstr
external_field_terms.f   | lowcase
external_field_terms.f   | strip
force_drivers.f          | coul0
force_drivers.f          | coul0neu
force_drivers.f          | coul2
force_drivers.f          | coul2neu
force_drivers.f          | coul3
force_drivers.f          | coul3neu
force_drivers.f          | coul4
force_drivers.f          | erfcgen
force_drivers.f          | error
force_drivers.f          | ewald1
force_drivers.f          | ewald2
force_drivers.f          | ewald3
force_drivers.f          | ewald4
force_drivers.f          | ewald_spme
force_drivers.f          | gdsum
force_drivers.f          | gimax
```

```
force_drivers.f          | gstate
force_drivers.f          | hkewald1
force_drivers.f          | hkewald2
force_drivers.f          | hkewald3
force_drivers.f          | hkewald4
force_drivers.f          | hkgen
force_drivers.f          | images
force_drivers.f          | metdens
force_drivers.f          | metfrc
force_drivers.f          | neutlst
force_drivers.f          | primlst
force_drivers.f          | prneulst
force_drivers.f          | rdf0
force_drivers.f          | rdf0neu
force_drivers.f          | srfrce
force_drivers.f          | srfrceneu
four_body_module.f       | error
four_body_terms.f        | copystring
four_body_terms.f        | dblstr
four_body_terms.f        | dcell
four_body_terms.f        | error
four_body_terms.f        | gdsum
four_body_terms.f        | getrec
four_body_terms.f        | getword
four_body_terms.f        | gstate
four_body_terms.f        | intstr
four_body_terms.f        | invert
four_body_terms.f        | lowcase
hkewald_module.f         | error
hkewald_terms.f          | cerfr
hkewald_terms.f          | dcell
hkewald_terms.f          | error
hkewald_terms.f          | gdsum
hkewald_terms.f          | invert
hkewald_terms.f          | warning
inversion_module.f       | error
inversion_terms.f        | copystring
inversion_terms.f        | dblstr
inversion_terms.f        | error
inversion_terms.f        | gdsum
inversion_terms.f        | getrec
inversion_terms.f        | getword
inversion_terms.f        | gstate
inversion_terms.f        | images
```

```
inversion_terms.f       | intstr
inversion_terms.f       | lowcase
kinetic_terms.f         | gdsum
lf_integrate.f          | error
lf_integrate.f          | gstate
lf_integrate.f          | npt_b1
lf_integrate.f          | npt_h1
lf_integrate.f          | nptq_b1
lf_integrate.f          | nptq_b2
lf_integrate.f          | nptq_h1
lf_integrate.f          | nptq_h2
lf_integrate.f          | nst_b1
lf_integrate.f          | nst_h1
lf_integrate.f          | nstq_b1
lf_integrate.f          | nstq_b2
lf_integrate.f          | nstq_h1
lf_integrate.f          | nstq_h2
lf_integrate.f          | nve_1
lf_integrate.f          | nveq_1
lf_integrate.f          | nveq_2
lf_integrate.f          | nvt_b1
lf_integrate.f          | nvt_e1
lf_integrate.f          | nvt_h1
lf_integrate.f          | nvtq_b1
lf_integrate.f          | nvtq_b2
lf_integrate.f          | nvtq_h1
lf_integrate.f          | nvtq_h2
lf_integrate.f          | pmflf
lf_motion_1.f           | cell_propagate
lf_motion_1.f           | error
lf_motion_1.f           | gdsum
lf_motion_1.f           | getcom
lf_motion_1.f           | getkin
lf_motion_1.f           | getmass
lf_motion_1.f           | getvom
lf_motion_1.f           | gstate
lf_motion_1.f           | images
lf_motion_1.f           | kinstress
lf_motion_1.f           | matmul
lf_motion_1.f           | merge
lf_motion_1.f           | rdshake_1
lf_motion_1.f           | sdot0
lf_motion_1.f           | shmove
lf_motion_1.f           | splice
```

```
lf_rotation_1.f          | bodystress
lf_rotation_1.f          | cell_propagate
lf_rotation_1.f          | error
lf_rotation_1.f          | getcom
lf_rotation_1.f          | getkinf
lf_rotation_1.f          | getkinr
lf_rotation_1.f          | getkint
lf_rotation_1.f          | getmass
lf_rotation_1.f          | getrotmat
lf_rotation_1.f          | getvom
lf_rotation_1.f          | gimax
lf_rotation_1.f          | gstate
lf_rotation_1.f          | images
lf_rotation_1.f          | kinstressf
lf_rotation_1.f          | kinstressg
lf_rotation_1.f          | matmul
lf_rotation_1.f          | merge
lf_rotation_1.f          | merge1
lf_rotation_1.f          | rdshake_1
lf_rotation_1.f          | sdot0
lf_rotation_1.f          | update_quaternions
lf_rotation_2.f          | bodystress
lf_rotation_2.f          | cell_propagate
lf_rotation_2.f          | error
lf_rotation_2.f          | gdsum
lf_rotation_2.f          | getcom
lf_rotation_2.f          | getkinf
lf_rotation_2.f          | getkinr
lf_rotation_2.f          | getkint
lf_rotation_2.f          | getmass
lf_rotation_2.f          | getrotmat
lf_rotation_2.f          | getvom
lf_rotation_2.f          | gimax
lf_rotation_2.f          | gstate
lf_rotation_2.f          | images
lf_rotation_2.f          | kinstressf
lf_rotation_2.f          | kinstressg
lf_rotation_2.f          | matmul
lf_rotation_2.f          | merge
lf_rotation_2.f          | merge1
lf_rotation_2.f          | merge4
lf_rotation_2.f          | qshake
lf_rotation_2.f          | shmove
lf_rotation_2.f          | splice
```

```
lf_rotation_2.f          | update_quaternions
merge_hcube.f            | nodedim
merge_systol.f           | nodedim
merge_tools.f            | MPI_IRECV
merge_tools.f            | MPI_SEND
merge_tools.f            | MPI_WAIT
merge_tools.f            | error
merge_tools.f            | gdsum
merge_tools.f            | gstate
merge_tools.f            | gsync
metal_module.f           | error
metal_terms.f            | abort_eamtable_read
metal_terms.f            | copystring
metal_terms.f            | dblstr
metal_terms.f            | eamden
metal_terms.f            | error
metal_terms.f            | findstring
metal_terms.f            | fsden
metal_terms.f            | gdsum
metal_terms.f            | getrec
metal_terms.f            | getword
metal_terms.f            | images
metal_terms.f            | intstr
metal_terms.f            | loc8
metal_terms.f            | lowcase
metal_terms.f            | metal_deriv
metal_terms.f            | metgen
metal_terms.f            | mettab
metal_terms.f            | warning
nlist_builders.f         | dcell
nlist_builders.f         | error
nlist_builders.f         | gimax
nlist_builders.f         | gisum
nlist_builders.f         | gstate
nlist_builders.f         | images
nlist_builders.f         | invert
nlist_builders.f         | merge
nlist_builders.f         | shellsort
pair_module.f            | error
parse_tools.f            | gisum
parse_tools.f            | gstate
parse_tools.f            | gsync
pass_tools.f             | MPI_IRECV
pass_tools.f             | MPI_SEND
```

```
pass_tools.f            | MPI_WAIT
pass_tools.f            | error
pass_tools.f            | gisum
pass_tools.f            | gstate
pass_tools.f            | gsync
pmf_lf.f                | error
pmf_lf.f                | gdsum
pmf_lf.f                | getkin
pmf_lf.f                | images
pmf_lf.f                | kinstress
pmf_lf.f                | merge
pmf_lf.f                | pmf_shake
pmf_lf.f                | pmf_vectors
pmf_lf.f                | rdshake_1
pmf_lf.f                | splice
pmf_module.f            | error
pmf_terms.f             | dblstr
pmf_terms.f             | error
pmf_terms.f             | findstring
pmf_terms.f             | getrec
pmf_terms.f             | images
pmf_terms.f             | intstr
pmf_terms.f             | lowcase
pmf_terms.f             | strip
pmf_vv.f                | error
pmf_vv.f                | gdsum
pmf_vv.f                | getkin
pmf_vv.f                | images
pmf_vv.f                | kinstress
pmf_vv.f                | merge
pmf_vv.f                | pmf_rattle_v
pmf_vv.f                | pmf_vectors
pmf_vv.f                | rdrattle_r
pmf_vv.f                | rdrattle_v
pmf_vv.f                | splice
property_module.f       | error
rigid_body_module.f     | error
rigid_body_terms.f      | error
rigid_body_terms.f      | gdsum
rigid_body_terms.f      | getrec
rigid_body_terms.f      | intstr
serial.f                | error
setup_program.f         | abortscan
setup_program.f         | cfgscan
```

```
setup_program.f          | conscan
setup_program.f          | dblstr
setup_program.f          | dcell
setup_program.f          | error
setup_program.f          | findstring
setup_program.f          | fldscan
setup_program.f          | gdsum
setup_program.f          | getrec
setup_program.f          | getword
setup_program.f          | intstr
setup_program.f          | lowcase
shake_module.f           | error
shake_terms.f            | copystring
shake_terms.f            | dblstr
shake_terms.f            | error
shake_terms.f            | getrec
shake_terms.f            | intstr
site_module.f            | error
site_terms.f             | copystring
site_terms.f             | dblstr
site_terms.f             | error
site_terms.f             | getrec
site_terms.f             | getword
site_terms.f             | intstr
site_terms.f             | warning
spme_module.f            | error
spme_terms.f             | bspcoe
spme_terms.f             | bspgen
spme_terms.f             | ccfft3d
spme_terms.f             | cpy_rtc
spme_terms.f             | dcell
spme_terms.f             | dcft3
spme_terms.f             | dlpfft3
spme_terms.f             | ele_prd
spme_terms.f             | error
spme_terms.f             | fftwnd_f77_one
spme_terms.f             | gdsum
spme_terms.f             | invert
spme_terms.f             | scl_csum
spme_terms.f             | set_block
spme_terms.f             | spl_cexp
spme_terms.f             | spme_for
spme_terms.f             | zzfft3d
strucopt.f               | error
```

```
strucopt.f              | images
system_properties.f     | dcell
system_properties.f     | diffsn0
system_properties.f     | diffsn1
system_properties.f     | error
system_properties.f     | gdsum
system_properties.f     | merge
system_properties.f     | rdf1
system_properties.f     | revive
system_properties.f     | timchk
system_properties.f     | zden1
temp_scalers.f          | error
temp_scalers.f          | gdsum
temp_scalers.f          | gstate
temp_scalers.f          | images
temp_scalers.f          | invert
temp_scalers.f          | merge
temp_scalers.f          | merge1
temp_scalers.f          | quatqnch
temp_scalers.f          | shlmerge
temp_scalers.f          | shmove
temp_scalers.f          | splice
tersoff_module.f        | error
tersoff_terms.f         | copystring
tersoff_terms.f         | dblstr
tersoff_terms.f         | dcell
tersoff_terms.f         | error
tersoff_terms.f         | gdsum
tersoff_terms.f         | getrec
tersoff_terms.f         | getword
tersoff_terms.f         | gstate
tersoff_terms.f         | intstr
tersoff_terms.f         | invert
tersoff_terms.f         | loc8
tersoff_terms.f         | lowcase
tersoff_terms.f         | tergen
tersoff_terms.f         | terint
tersoff_terms.f         | tersoff3
tether_module.f         | error
tether_terms.f          | copystring
tether_terms.f          | dblstr
tether_terms.f          | error
tether_terms.f          | gdsum
tether_terms.f          | getrec
```

```
tether_terms.f          | getword
tether_terms.f          | gstate
tether_terms.f          | images
tether_terms.f          | intstr
tether_terms.f          | lowcase
tether_terms.f          | strip
three_body_module.f     | error
three_body_terms.f      | copystring
three_body_terms.f      | dblstr
three_body_terms.f      | dcell
three_body_terms.f      | error
three_body_terms.f      | gdsum
three_body_terms.f      | getrec
three_body_terms.f      | getword
three_body_terms.f      | gstate
three_body_terms.f      | intstr
three_body_terms.f      | invert
three_body_terms.f      | lowcase
timchk.f                | date_and_time
timchk.f                | mynode
utility_pack.f          | date_and_time
utility_pack.f          | error
utility_pack.f          | exitcomms
utility_pack.f          | gdsum
utility_pack.f          | images
utility_pack.f          | intstr3
utility_pack.f          | invert
utility_pack.f          | merge
utility_pack.f          | sdot0
utility_pack.f          | sdot1
vdw_module.f            | error
vdw_terms.f             | abort_table_read
vdw_terms.f             | copystring
vdw_terms.f             | dblstr
vdw_terms.f             | error
vdw_terms.f             | findstring
vdw_terms.f             | forgen
vdw_terms.f             | fortab
vdw_terms.f             | gdsum
vdw_terms.f             | getrec
vdw_terms.f             | getword
vdw_terms.f             | intstr
vdw_terms.f             | loc8
vdw_terms.f             | lowcase
```

```
vdw_terms.f             | warning
vv_integrate.f          | error
vv_integrate.f          | gstate
vv_integrate.f          | nptqvv_b1
vv_integrate.f          | nptqvv_b2
vv_integrate.f          | nptqvv_h1
vv_integrate.f          | nptqvv_h2
vv_integrate.f          | nptvv_b1
vv_integrate.f          | nptvv_h1
vv_integrate.f          | nstqvv_b1
vv_integrate.f          | nstqvv_b2
vv_integrate.f          | nstqvv_h1
vv_integrate.f          | nstqvv_h2
vv_integrate.f          | nstvv_b1
vv_integrate.f          | nstvv_h1
vv_integrate.f          | nveqvv_1
vv_integrate.f          | nveqvv_2
vv_integrate.f          | nvevv_1
vv_integrate.f          | nvtqvv_b1
vv_integrate.f          | nvtqvv_b2
vv_integrate.f          | nvtqvv_h1
vv_integrate.f          | nvtqvv_h2
vv_integrate.f          | nvtvv_b1
vv_integrate.f          | nvtvv_e1
vv_integrate.f          | nvtvv_h1
vv_integrate.f          | pmfvv
vv_motion_1.f           | dcell
vv_motion_1.f           | error
vv_motion_1.f           | gdsum
vv_motion_1.f           | getcom
vv_motion_1.f           | getkin
vv_motion_1.f           | getmass
vv_motion_1.f           | getvom
vv_motion_1.f           | gstate
vv_motion_1.f           | images
vv_motion_1.f           | kinstress
vv_motion_1.f           | matmul
vv_motion_1.f           | merge
vv_motion_1.f           | nptscale_p
vv_motion_1.f           | nptscale_t
vv_motion_1.f           | nstscale_p
vv_motion_1.f           | nstscale_t
vv_motion_1.f           | nvtscale
vv_motion_1.f           | rdrattle_r
```

```
vv_motion_1.f           | rdrattle_v
vv_motion_1.f           | sdot0
vv_motion_1.f           | shmove
vv_motion_1.f           | splice
vv_rotation_1.f         | bodystress
vv_rotation_1.f         | dcell
vv_rotation_1.f         | error
vv_rotation_1.f         | getcom
vv_rotation_1.f         | getkinf
vv_rotation_1.f         | getking
vv_rotation_1.f         | getking
vv_rotation_1.f         | getmass
vv_rotation_1.f         | getrotmat
vv_rotation_1.f         | getvom
vv_rotation_1.f         | gimax
vv_rotation_1.f         | gstate
vv_rotation_1.f         | images
vv_rotation_1.f         | kinstressf
vv_rotation_1.f         | kinstressg
vv_rotation_1.f         | matmul
vv_rotation_1.f         | merge
vv_rotation_1.f         | merge1
vv_rotation_1.f         | nosquish
vv_rotation_1.f         | nptqscl_p
vv_rotation_1.f         | nptqscl_t
vv_rotation_1.f         | nstqscl_p
vv_rotation_1.f         | nstqscl_t
vv_rotation_1.f         | nvtqscl
vv_rotation_1.f         | rdrattle_r
vv_rotation_1.f         | rdrattle_v
vv_rotation_1.f         | sdot0
vv_rotation_2.f         | bodystress
vv_rotation_2.f         | dcell
vv_rotation_2.f         | error
vv_rotation_2.f         | gdsum
vv_rotation_2.f         | getcom
vv_rotation_2.f         | getkinf
vv_rotation_2.f         | getking
vv_rotation_2.f         | getking
vv_rotation_2.f         | getmass
vv_rotation_2.f         | getrotmat
vv_rotation_2.f         | getvom
vv_rotation_2.f         | gimax
vv_rotation_2.f         | gstate
```

```
vv_rotation_2.f          | images
vv_rotation_2.f          | kinstressf
vv_rotation_2.f          | kinstressg
vv_rotation_2.f          | matmul
vv_rotation_2.f          | merge
vv_rotation_2.f          | merge1
vv_rotation_2.f          | nosquish
vv_rotation_2.f          | nptqscl_p
vv_rotation_2.f          | nptqscl_t
vv_rotation_2.f          | nstqscl_p2
vv_rotation_2.f          | nstqscl_t2
vv_rotation_2.f          | nvtqscl
vv_rotation_2.f          | pivot
vv_rotation_2.f          | qrattle_r
vv_rotation_2.f          | qrattle_v
vv_rotation_2.f          | rotate_omega
vv_rotation_2.f          | sdot0
vv_rotation_2.f          | shmove
vv_rotation_2.f          | splice
```

Appendix F

Calling Subroutines

Calling Subroutines

The following table lists the subroutines in DL_POLY_2 and where they are called from.

Called routine	Calling routine
abort_config_read	define_system.f
abort_control_read	define_system.f
abort_eamtable_read	metal_terms.f
abort_field_read	define_system.f
abort_table_read	vdw_terms.f
abortscan	setup_program.f
alloc_ang_arrays	dlpoly.f
alloc_bnd_arrays	dlpoly.f
alloc_config_arrays	dlpoly.f
alloc_csh_arrays	dlpoly.f
alloc_dih_arrays	dlpoly.f
alloc_ewald_arrays	dlpoly.f
alloc_exc_arrays	dlpoly.f
alloc_fbp_arrays	dlpoly.f
alloc_fld_arrays	dlpoly.f
alloc_hke_arrays	dlpoly.f
alloc_inv_arrays	dlpoly.f
alloc_met_arrays	dlpoly.f
alloc_pair_arrays	dlpoly.f
alloc_pmf_arrays	dlpoly.f
alloc_prp_arrays	dlpoly.f
alloc_rgbdy_arrays	dlpoly.f
alloc_shake_arrays	dlpoly.f
alloc_site_arrays	dlpoly.f
alloc_spme_arrays	dlpoly.f
alloc_tbp_arrays	dlpoly.f

```
alloc_ter_arrays        | dlpoly.f
alloc_tet_arrays        | dlpoly.f
alloc_vdw_arrays        | dlpoly.f
angfrc                  | dlpoly.f
bndfrc                  | dlpoly.f
bodystress              | lf_rotation_1.f
bodystress              | lf_rotation_2.f
bodystress              | vv_rotation_1.f
bodystress              | vv_rotation_2.f
bspcoe                  | spme_terms.f
bspgen                  | spme_terms.f
ccfft3d                 | define_system.f
ccfft3d                 | spme_terms.f
cell_propagate          | lf_motion_1.f
cell_propagate          | lf_rotation_1.f
cell_propagate          | lf_rotation_2.f
cell_update             | ensemble_tools.f
cerfr                   | hkewald_terms.f
cfgscan                 | setup_program.f
check_shells            | define_system.f
check_syschg            | define_system.f
conscan                 | setup_program.f
copystring              | angle_terms.f
copystring              | bond_terms.f
copystring              | define_system.f
copystring              | dihedral_terms.f
copystring              | external_field_terms.f
copystring              | four_body_terms.f
copystring              | inversion_terms.f
copystring              | metal_terms.f
copystring              | shake_terms.f
copystring              | site_terms.f
copystring              | tersoff_terms.f
copystring              | tether_terms.f
copystring              | three_body_terms.f
copystring              | vdw_terms.f
corshl                  | dlpoly.f
coul0                   | force_drivers.f
coul0neu                | force_drivers.f
coul2                   | force_drivers.f
coul2neu                | force_drivers.f
coul3                   | force_drivers.f
coul3neu                | force_drivers.f
coul4                   | force_drivers.f
```

```
cpy_rtc                 | spme_terms.f
date_and_time           | timchk.f
date_and_time           | utility_pack.f
dblstr                  | angle_terms.f
dblstr                  | bond_terms.f
dblstr                  | core_shell_terms.f
dblstr                  | define_system.f
dblstr                  | dihedral_terms.f
dblstr                  | external_field_terms.f
dblstr                  | four_body_terms.f
dblstr                  | inversion_terms.f
dblstr                  | metal_terms.f
dblstr                  | pmf_terms.f
dblstr                  | setup_program.f
dblstr                  | shake_terms.f
dblstr                  | site_terms.f
dblstr                  | tersoff_terms.f
dblstr                  | tether_terms.f
dblstr                  | three_body_terms.f
dblstr                  | vdw_terms.f
dcell                   | define_system.f
dcell                   | dlpoly.f
dcell                   | ensemble_tools.f
dcell                   | ewald_terms.f
dcell                   | four_body_terms.f
dcell                   | hkewald_terms.f
dcell                   | nlist_builders.f
dcell                   | setup_program.f
dcell                   | spme_terms.f
dcell                   | system_properties.f
dcell                   | tersoff_terms.f
dcell                   | three_body_terms.f
dcell                   | vv_motion_1.f
dcell                   | vv_rotation_1.f
dcell                   | vv_rotation_2.f
dcft3                   | spme_terms.f
define_angles           | define_system.f
define_atoms            | define_system.f
define_bonds            | define_system.f
define_constraints      | define_system.f
define_core_shell       | define_system.f
define_dihedrals        | define_system.f
define_external_field   | define_system.f
define_four_body        | define_system.f
```

```
define_inversions        | define_system.f
define_metals            | define_system.f
define_pmf               | define_system.f
define_rigid_body        | define_system.f
define_tersoff           | define_system.f
define_tethers           | define_system.f
define_three_body        | define_system.f
define_units             | define_system.f
define_van_der_waals     | define_system.f
diffsn0                  | system_properties.f
diffsn1                  | system_properties.f
dihfrc                   | dlpoly.f
dlpfft3                  | spme_terms.f
eamden                   | metal_terms.f
ele_prd                  | spme_terms.f
erfcgen                  | force_drivers.f
error                    | angle_terms.f
error                    | angles_module.f
error                    | bond_terms.f
error                    | bonds_module.f
error                    | config_module.f
error                    | core_shell_module.f
error                    | core_shell_terms.f
error                    | define_system.f
error                    | dihed_module.f
error                    | dihedral_terms.f
error                    | dlpoly.f
error                    | ewald_module.f
error                    | ewald_terms.f
error                    | exclude_module.f
error                    | exclude_terms.f
error                    | external_field_module.f
error                    | external_field_terms.f
error                    | force_drivers.f
error                    | four_body_module.f
error                    | four_body_terms.f
error                    | hkewald_module.f
error                    | hkewald_terms.f
error                    | inversion_module.f
error                    | inversion_terms.f
error                    | lf_integrate.f
error                    | lf_motion_1.f
error                    | lf_rotation_1.f
error                    | lf_rotation_2.f
```

```
error                | merge_tools.f
error                | metal_module.f
error                | metal_terms.f
error                | nlist_builders.f
error                | pair_module.f
error                | pass_tools.f
error                | pmf_lf.f
error                | pmf_module.f
error                | pmf_terms.f
error                | pmf_vv.f
error                | property_module.f
error                | rigid_body_module.f
error                | rigid_body_terms.f
error                | serial.f
error                | setup_program.f
error                | shake_module.f
error                | shake_terms.f
error                | site_module.f
error                | site_terms.f
error                | spme_module.f
error                | spme_terms.f
error                | strucopt.f
error                | system_properties.f
error                | temp_scalers.f
error                | tersoff_module.f
error                | tersoff_terms.f
error                | tether_module.f
error                | tether_terms.f
error                | three_body_module.f
error                | three_body_terms.f
error                | utility_pack.f
error                | vdw_module.f
error                | vdw_terms.f
error                | vv_integrate.f
error                | vv_motion_1.f
error                | vv_rotation_1.f
error                | vv_rotation_2.f
ewald1               | force_drivers.f
ewald2               | force_drivers.f
ewald3               | force_drivers.f
ewald4               | force_drivers.f
ewald_spme           | force_drivers.f
exclude              | define_system.f
exclude_atom         | define_system.f
```

```
exclude_link              | define_system.f
excludeneu                | define_system.f
exit                      | basic_comms.f
exitcomms                 | dlpoly.f
exitcomms                 | error.f
exitcomms                 | utility_pack.f
extnfld                   | dlpoly.f
fbpfrc                    | dlpoly.f
fcap                      | dlpoly.f
fftw3d_f77_create_plan    | define_system.f
fftwnd_f77_one            | spme_terms.f
findstring                | define_system.f
findstring                | metal_terms.f
findstring                | pmf_terms.f
findstring                | setup_program.f
findstring                | vdw_terms.f
fldscan                   | setup_program.f
forces                    | dlpoly.f
forcesneu                 | dlpoly.f
forgen                    | vdw_terms.f
fortab                    | vdw_terms.f
freeze                    | dlpoly.f
fsden                     | metal_terms.f
gauss                     | define_system.f
gdsum                     | angle_terms.f
gdsum                     | bond_terms.f
gdsum                     | core_shell_terms.f
gdsum                     | define_system.f
gdsum                     | dihedral_terms.f
gdsum                     | dlpoly.f
gdsum                     | ensemble_tools.f
gdsum                     | ewald_terms.f
gdsum                     | force_drivers.f
gdsum                     | four_body_terms.f
gdsum                     | hkewald_terms.f
gdsum                     | inversion_terms.f
gdsum                     | kinetic_terms.f
gdsum                     | lf_motion_1.f
gdsum                     | lf_rotation_2.f
gdsum                     | merge_tools.f
gdsum                     | metal_terms.f
gdsum                     | pmf_lf.f
gdsum                     | pmf_vv.f
gdsum                     | rigid_body_terms.f
```

```
gdsum                | setup_program.f
gdsum                | spme_terms.f
gdsum                | system_properties.f
gdsum                | temp_scalers.f
gdsum                | tersoff_terms.f
gdsum                | tether_terms.f
gdsum                | three_body_terms.f
gdsum                | utility_pack.f
gdsum                | vdw_terms.f
gdsum                | vv_motion_1.f
gdsum                | vv_rotation_2.f
getcom               | lf_motion_1.f
getcom               | lf_rotation_1.f
getcom               | lf_rotation_2.f
getcom               | vv_motion_1.f
getcom               | vv_rotation_1.f
getcom               | vv_rotation_2.f
getkin               | ensemble_tools.f
getkin               | lf_motion_1.f
getkin               | pmf_lf.f
getkin               | pmf_vv.f
getkin               | vv_motion_1.f
getkinf              | ensemble_tools.f
getkinf              | lf_rotation_1.f
getkinf              | lf_rotation_2.f
getkinf              | vv_rotation_1.f
getkinf              | vv_rotation_2.f
getking              | ensemble_tools.f
getking              | ensemble_tools.f
getking              | vv_rotation_1.f
getking              | vv_rotation_1.f
getking              | vv_rotation_2.f
getking              | vv_rotation_2.f
getkinr              | lf_rotation_1.f
getkinr              | lf_rotation_2.f
getkint              | lf_rotation_1.f
getkint              | lf_rotation_2.f
getmass              | lf_motion_1.f
getmass              | lf_rotation_1.f
getmass              | lf_rotation_2.f
getmass              | vv_motion_1.f
getmass              | vv_rotation_1.f
getmass              | vv_rotation_2.f
getrec               | angle_terms.f
```

```
getrec                  | bond_terms.f
getrec                  | core_shell_terms.f
getrec                  | define_system.f
getrec                  | dihedral_terms.f
getrec                  | external_field_terms.f
getrec                  | four_body_terms.f
getrec                  | inversion_terms.f
getrec                  | metal_terms.f
getrec                  | pmf_terms.f
getrec                  | rigid_body_terms.f
getrec                  | setup_program.f
getrec                  | shake_terms.f
getrec                  | site_terms.f
getrec                  | tersoff_terms.f
getrec                  | tether_terms.f
getrec                  | three_body_terms.f
getrec                  | vdw_terms.f
getrotmat               | lf_rotation_1.f
getrotmat               | lf_rotation_2.f
getrotmat               | vv_rotation_1.f
getrotmat               | vv_rotation_2.f
getvom                  | lf_motion_1.f
getvom                  | lf_rotation_1.f
getvom                  | lf_rotation_2.f
getvom                  | vv_motion_1.f
getvom                  | vv_rotation_1.f
getvom                  | vv_rotation_2.f
getword                 | angle_terms.f
getword                 | bond_terms.f
getword                 | dihedral_terms.f
getword                 | external_field_terms.f
getword                 | four_body_terms.f
getword                 | inversion_terms.f
getword                 | metal_terms.f
getword                 | setup_program.f
getword                 | site_terms.f
getword                 | tersoff_terms.f
getword                 | tether_terms.f
getword                 | three_body_terms.f
getword                 | vdw_terms.f
gimax                   | define_system.f
gimax                   | exclude_terms.f
gimax                   | force_drivers.f
gimax                   | lf_rotation_1.f
```

```
gimax                   | lf_rotation_2.f
gimax                   | nlist_builders.f
gimax                   | vv_rotation_1.f
gimax                   | vv_rotation_2.f
gisum                   | basic_comms.f
gisum                   | nlist_builders.f
gisum                   | parse_tools.f
gisum                   | pass_tools.f
global_sum_forces       | dlpoly.f
gstate                  | angle_terms.f
gstate                  | bond_terms.f
gstate                  | define_system.f
gstate                  | dihedral_terms.f
gstate                  | exclude_terms.f
gstate                  | force_drivers.f
gstate                  | four_body_terms.f
gstate                  | inversion_terms.f
gstate                  | lf_integrate.f
gstate                  | lf_motion_1.f
gstate                  | lf_rotation_1.f
gstate                  | lf_rotation_2.f
gstate                  | merge_tools.f
gstate                  | nlist_builders.f
gstate                  | parse_tools.f
gstate                  | pass_tools.f
gstate                  | temp_scalers.f
gstate                  | tersoff_terms.f
gstate                  | tether_terms.f
gstate                  | three_body_terms.f
gstate                  | vv_integrate.f
gstate                  | vv_motion_1.f
gstate                  | vv_rotation_1.f
gstate                  | vv_rotation_2.f
gsync                   | dlpoly.f
gsync                   | error.f
gsync                   | merge_tools.f
gsync                   | parse_tools.f
gsync                   | pass_tools.f
hkewald1                | force_drivers.f
hkewald2                | force_drivers.f
hkewald3                | force_drivers.f
hkewald4                | force_drivers.f
hkgen                   | force_drivers.f
images                  | angle_terms.f
```

```
images                  | bond_terms.f
images                  | core_shell_terms.f
images                  | define_system.f
images                  | dihedral_terms.f
images                  | force_drivers.f
images                  | inversion_terms.f
images                  | lf_motion_1.f
images                  | lf_rotation_1.f
images                  | lf_rotation_2.f
images                  | metal_terms.f
images                  | nlist_builders.f
images                  | pmf_lf.f
images                  | pmf_terms.f
images                  | pmf_vv.f
images                  | strucopt.f
images                  | temp_scalers.f
images                  | tether_terms.f
images                  | utility_pack.f
images                  | vv_motion_1.f
images                  | vv_rotation_1.f
images                  | vv_rotation_2.f
initcomms               | dlpoly.f
intlist                 | define_system.f
intstr                  | angle_terms.f
intstr                  | bond_terms.f
intstr                  | core_shell_terms.f
intstr                  | define_system.f
intstr                  | dihedral_terms.f
intstr                  | external_field_terms.f
intstr                  | four_body_terms.f
intstr                  | inversion_terms.f
intstr                  | metal_terms.f
intstr                  | pmf_terms.f
intstr                  | rigid_body_terms.f
intstr                  | setup_program.f
intstr                  | shake_terms.f
intstr                  | site_terms.f
intstr                  | tersoff_terms.f
intstr                  | tether_terms.f
intstr                  | three_body_terms.f
intstr                  | vdw_terms.f
intstr3                 | utility_pack.f
invert                  | define_system.f
invert                  | ensemble_tools.f
```

```
invert                  | ewald_terms.f
invert                  | four_body_terms.f
invert                  | hkewald_terms.f
invert                  | nlist_builders.f
invert                  | spme_terms.f
invert                  | temp_scalers.f
invert                  | tersoff_terms.f
invert                  | three_body_terms.f
invert                  | utility_pack.f
invfrc                  | dlpoly.f
jacobi                  | define_system.f
kinstress               | dlpoly.f
kinstress               | ensemble_tools.f
kinstress               | lf_motion_1.f
kinstress               | pmf_lf.f
kinstress               | pmf_vv.f
kinstress               | vv_motion_1.f
kinstressf              | ensemble_tools.f
kinstressf              | lf_rotation_1.f
kinstressf              | lf_rotation_2.f
kinstressf              | vv_rotation_1.f
kinstressf              | vv_rotation_2.f
kinstressg              | ensemble_tools.f
kinstressg              | lf_rotation_1.f
kinstressg              | lf_rotation_2.f
kinstressg              | vv_rotation_1.f
kinstressg              | vv_rotation_2.f
lf_integrate            | dlpoly.f
loc8                    | metal_terms.f
loc8                    | tersoff_terms.f
loc8                    | vdw_terms.f
lowcase                 | angle_terms.f
lowcase                 | bond_terms.f
lowcase                 | define_system.f
lowcase                 | dihedral_terms.f
lowcase                 | external_field_terms.f
lowcase                 | four_body_terms.f
lowcase                 | inversion_terms.f
lowcase                 | metal_terms.f
lowcase                 | pmf_terms.f
lowcase                 | setup_program.f
lowcase                 | tersoff_terms.f
lowcase                 | tether_terms.f
lowcase                 | three_body_terms.f
```

```
lowcase                 | vdw_terms.f
lrcmetal                | define_system.f
lrcmetal                | dlpoly.f
lrcorrect               | define_system.f
machine                 | dlpoly.f
matmul                  | ensemble_tools.f
matmul                  | lf_motion_1.f
matmul                  | lf_rotation_1.f
matmul                  | lf_rotation_2.f
matmul                  | vv_motion_1.f
matmul                  | vv_rotation_1.f
matmul                  | vv_rotation_2.f
merge                   | core_shell_terms.f
merge                   | define_system.f
merge                   | lf_motion_1.f
merge                   | lf_rotation_1.f
merge                   | lf_rotation_2.f
merge                   | nlist_builders.f
merge                   | pmf_lf.f
merge                   | pmf_vv.f
merge                   | system_properties.f
merge                   | temp_scalers.f
merge                   | utility_pack.f
merge                   | vv_motion_1.f
merge                   | vv_rotation_1.f
merge                   | vv_rotation_2.f
merge1                  | core_shell_terms.f
merge1                  | lf_rotation_1.f
merge1                  | lf_rotation_2.f
merge1                  | temp_scalers.f
merge1                  | vv_rotation_1.f
merge1                  | vv_rotation_2.f
merge4                  | define_system.f
merge4                  | lf_rotation_2.f
metal_deriv             | metal_terms.f
metdens                 | force_drivers.f
metfrc                  | force_drivers.f
metgen                  | metal_terms.f
mettab                  | metal_terms.f
mkwd8                   | define_system.f
MPI_BARRIER             | basic_comms.f
MPI_COMM_RANK           | basic_comms.f
MPI_COMM_SIZE           | basic_comms.f
MPI_FINALIZE            | basic_comms.f
```

MPI_IRECV	merge_tools.f
MPI_IRECV	pass_tools.f
MPI_RECV	basic_comms.f
MPI_SEND	merge_tools.f
MPI_SEND	pass_tools.f
MPI_WAIT	merge_tools.f
MPI_WAIT	pass_tools.f
MPI_allreduce	basic_comms.f
MPI_init	basic_comms.f
MPI_send	basic_comms.f
multiple	dlpoly.f
multiple_nsq	dlpoly.f
multipleneu	dlpoly.f
mynode	basic_comms.f
mynode	timchk.f
neutbook	define_system.f
neutlst	force_drivers.f
nodedim	merge_hcube.f
nodedim	merge_systol.f
nosquish	vv_rotation_1.f
nosquish	vv_rotation_2.f
npt_b1	lf_integrate.f
npt_h1	lf_integrate.f
nptq_b1	lf_integrate.f
nptq_b2	lf_integrate.f
nptq_h1	lf_integrate.f
nptq_h2	lf_integrate.f
nptqscl_p	vv_rotation_1.f
nptqscl_p	vv_rotation_2.f
nptqscl_t	vv_rotation_1.f
nptqscl_t	vv_rotation_2.f
nptqvv_b1	vv_integrate.f
nptqvv_b2	vv_integrate.f
nptqvv_h1	vv_integrate.f
nptqvv_h2	vv_integrate.f
nptscale_p	vv_motion_1.f
nptscale_t	vv_motion_1.f
nptvv_b1	vv_integrate.f
nptvv_h1	vv_integrate.f
nst_b1	lf_integrate.f
nst_h1	lf_integrate.f
nstq_b1	lf_integrate.f
nstq_b2	lf_integrate.f
nstq_h1	lf_integrate.f

```
nstq_h2                 | lf_integrate.f
nstqscl_p               | vv_rotation_1.f
nstqscl_p2              | vv_rotation_2.f
nstqscl_t               | vv_rotation_1.f
nstqscl_t2              | vv_rotation_2.f
nstqvv_b1               | vv_integrate.f
nstqvv_b2               | vv_integrate.f
nstqvv_h1               | vv_integrate.f
nstqvv_h2               | vv_integrate.f
nstscale_p              | vv_motion_1.f
nstscale_t              | vv_motion_1.f
nstvv_b1                | vv_integrate.f
nstvv_h1                | vv_integrate.f
numnodes                | basic_comms.f
nve_1                   | lf_integrate.f
nveq_1                  | lf_integrate.f
nveq_2                  | lf_integrate.f
nveqvv_1                | vv_integrate.f
nveqvv_2                | vv_integrate.f
nvevv_1                 | vv_integrate.f
nvt_b1                  | lf_integrate.f
nvt_e1                  | lf_integrate.f
nvt_h1                  | lf_integrate.f
nvtq_b1                 | lf_integrate.f
nvtq_b2                 | lf_integrate.f
nvtq_h1                 | lf_integrate.f
nvtq_h2                 | lf_integrate.f
nvtqscl                 | vv_rotation_1.f
nvtqscl                 | vv_rotation_2.f
nvtqvv_b1               | vv_integrate.f
nvtqvv_b2               | vv_integrate.f
nvtqvv_h1               | vv_integrate.f
nvtqvv_h2               | vv_integrate.f
nvtscale                | vv_motion_1.f
nvtvv_b1                | vv_integrate.f
nvtvv_e1                | vv_integrate.f
nvtvv_h1                | vv_integrate.f
parlink                 | dlpoly.f
parlinkneu              | dlpoly.f
parlst                  | dlpoly.f
parlst_nsq              | dlpoly.f
parneulst               | dlpoly.f
parset                  | dlpoly.f
passcon                 | define_system.f
```

passpmf	define_system.f
passquat	define_system.f
pivot	vv_rotation_2.f
pmf_rattle_v	pmf_vv.f
pmf_shake	pmf_lf.f
pmf_vectors	pmf_lf.f
pmf_vectors	pmf_vv.f
pmflf	lf_integrate.f
pmfvv	vv_integrate.f
primlst	force_drivers.f
prneulst	force_drivers.f
put_shells_on_cores	define_system.f
qrattle_r	vv_rotation_2.f
qrattle_v	vv_rotation_2.f
qshake	lf_rotation_2.f
quatbook	define_system.f
quatqnch	define_system.f
quatqnch	dlpoly.f
quatqnch	temp_scalers.f
quench	define_system.f
rdf0	force_drivers.f
rdf0neu	force_drivers.f
rdf1	system_properties.f
rdrattle_r	pmf_vv.f
rdrattle_r	vv_motion_1.f
rdrattle_r	vv_rotation_1.f
rdrattle_v	pmf_vv.f
rdrattle_v	vv_motion_1.f
rdrattle_v	vv_rotation_1.f
rdshake_1	lf_motion_1.f
rdshake_1	lf_rotation_1.f
rdshake_1	pmf_lf.f
relax_shells	dlpoly.f
result	dlpoly.f
revive	dlpoly.f
revive	system_properties.f
rotate_omega	vv_rotation_2.f
scl_csum	spme_terms.f
sdot0	ensemble_tools.f
sdot0	lf_motion_1.f
sdot0	lf_rotation_1.f
sdot0	utility_pack.f
sdot0	vv_motion_1.f
sdot0	vv_rotation_1.f

sdot0	vv_rotation_2.f
sdot1	utility_pack.f
set_block	spme_terms.f
shellsort	define_system.f
shellsort	nlist_builders.f
shlfrc	dlpoly.f
shlmerge	temp_scalers.f
shlqnch	define_system.f
shlqnch	dlpoly.f
shmove	lf_motion_1.f
shmove	lf_rotation_2.f
shmove	temp_scalers.f
shmove	vv_motion_1.f
shmove	vv_rotation_2.f
simdef	dlpoly.f
spl_cexp	spme_terms.f
splice	lf_motion_1.f
splice	lf_rotation_2.f
splice	pmf_lf.f
splice	pmf_vv.f
splice	temp_scalers.f
splice	vv_motion_1.f
splice	vv_rotation_2.f
spme_for	spme_terms.f
srfrce	force_drivers.f
srfrceneu	force_drivers.f
static	dlpoly.f
strip	define_system.f
strip	external_field_terms.f
strip	pmf_terms.f
strip	tether_terms.f
strucopt	dlpoly.f
sysbook	dlpoly.f
sysdef	dlpoly.f
sysgen	dlpoly.f
sysinit	dlpoly.f
systemp	dlpoly.f
tergen	tersoff_terms.f
terint	tersoff_terms.f
tersoff	dlpoly.f
tersoff3	tersoff_terms.f
tethfrc	dlpoly.f
thbfrc	dlpoly.f
timchk	dlpoly.f

```
timchk                  | system_properties.f
traject                 | dlpoly.f
update_quaternions      | lf_rotation_1.f
update_quaternions      | lf_rotation_2.f
vertest                 | dlpoly.f
vscaleg                 | define_system.f
vscaleg                 | dlpoly.f
vv_integrate            | dlpoly.f
warning                 | define_system.f
warning                 | exclude_terms.f
warning                 | hkewald_terms.f
warning                 | metal_terms.f
warning                 | site_terms.f
warning                 | vdw_terms.f
xscale                  | dlpoly.f
zden0                   | dlpoly.f
zden1                   | system_properties.f
zzfft3d                 | define_system.f
zzfft3d                 | spme_terms.f
```

Index