[19] S. Nosé and M. L. Klein. Constant pressure molecular dynamics for molecular systems. *Molecular Physics*, 50(5):1055–1076, 1983.

[20] B. Quentrec and C. Brot. New methods for searching for neighbours in molecular dynamics computations. *Journal of Computational Physics*, 13:430–432, 1975.

[21] D. M. Heyes. Large numbers of particles on the Cray-1. *Information Quarterly - CCP5*, 25:57–61, June 1987.

[22] M. Parinello and A. Rahman. Polymorphic transitions in single crystals: A new molecular dynamics method. *Journal of Applied Physics*, 52(12):7182–7190, 1981.

[23] J. P. Hansen and I. R. McDonald. *Theory of Simple Liquids*. Academic Press, London, 2nd edition, 1986.

[24] O. Matsouka, E. Clementi, and M. Yoshimine. CI study of the water dimer potential surface. *Journal of Chemical Physics*, 64(4):1351–1361, 1976.

[25] Sun Microsystems Inc. External data representation: Protocol specification. Technical Report RFC1014.

[26] D. Fincham. Optimisation of the ewald sum. *Information Quarterly - CCP5*, 38:17–24, August 1993.

[27] B. W. Kernighan and D. Ritchie. *The C programming language*. Prentice Hall, Cambridge, 1st edition, 1978.

[28] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, New York, 1970.

[29] M. Wojcik. MD simulation of MCY water using LCAP. Internal Report KGN-28, IBM Corporation, Kingston, N.Y.12401, USA, 1985.

[30] W. L. Jorgensen. Revised TIPS model for simulations of liquid water and aqueous solutions. *Journal of Chemical Physics*, 77:4156–63, 1982.

[31] W. Deitz, W. O. Riede, and K. Heinzinger. Molecular dynamics simulation of an aqueous $MgCl_2$ solution. *Z. Naturforsch*, 37a:1038–1048, 1982.

[32] H. Kistenmacher, H. Popkie, and E. Clementi. Study of the structure of molecular complexes III. Energy surface of a water molecule in the field of a fluorine or chlorine anion. *Journal of Chemical Physics*, 58(4):5842, 1973.

[33] E. Keller. Neues von SCHAKAL. *Chemie in unserer Zeit*, 20(6):178–181, 1986.

Bibliography

[1] M. P. Allen and D. J. Tildesley. *Computer simulation of liquids.* Clarendon Press, Oxford, 1987.

[2] D. J. Evans. On the representation of orientation space. *Molecular Physics*, 34(2):317–325, 1977.

[3] D. J. Evans and S. Murad. Singularity free algorithm for molecular dynamics simulation of rigid polyatomics. *Molecular Physics*, 34(2):327–331, 1977.

[4] G. S. Pawley and M. T. Dove. Quaternion-based reorientation conditions for molecular dynamics analyses. *Molecular Physics*, 55(5):1147–1157, 1985.

[5] G. S. Pawley. Molecular dynamics simulation of the plastic phase; a model for SF_6. *Molecular Physics*, 43(6):1321–1330, 1981.

[6] P. Du Val. *Homographies, Quaternions and Rotations.* Oxford Mathematical Monograph, 1964.

[7] R. Sonnenschein. An improved algorithm for molecular dynamics simulation of rigid molecules. *Journal of Computational Physics*, 59(2):347–350, 1985.

[8] W. Smith and D. Fincham. *The Program MDMPOL.* SERC, Daresbury Laboratory, 1982. CCP5 Program Library.

[9] A. Laakonsen. Computer simulation package for liquids and solids with polar interactions. I. McMOLDYN/H2O: aqueous systems. Internal Report KGN-41, IBM Corporation, Kingston, N.Y.12401, USA, 1985.

[10] H. Goldstein. *Classical Mechanics.* Addison–Wesley, Reading, MA, 2nd edition, 1980.

[11] K. Refson and G. S. Pawley. Molecular dynamics studies of the condensed phases of *n*-butane and their transitions, I Techniques and model results. *Molecular Physics*, 61(3):669–692, 1987.

[12] J. G. Powles, W. A. B. Evans, E. McGrath, K. E. Gubbins, and S. Murad. A computer simulation for a simple model of liquid hydrogen chloride. *Molecular Physics*, 38:893–908, 1979.

[13] K. Refson. Molecular dynamics simulation of solid *n*-butane. *Physica*, 131B:256–266, 1985.

[14] D. Beeman. Some multistep methods for use in molecular dynamics calculations. *Journal of Computational Physics*, 20:130–139, 1976.

[15] L. Verlet. Computer 'experiments' on classical fluids. I. thermodynamical properties of lennard-jones molecules. *Physical Review*, 165:201–214, 1967.

[16] D. Fincham. Rotational verlet. *Information Quarterly - CCP5*, 5:6, 1981.

[17] F. Berthaut. L'énergie électrostatique de rèseaux ioniques. *J. Phys. Radium*, 13:499–505, 1952.

[18] S. W. De Leeuw, J. W. Perram, and E. R. Smith. Simulation or electrostatic systems in periodic boundary conditions. I Lattice sums and dielectric constant. *Proceedings of the Royal Society*, A373:27–56, 1980.

-i this inserts its argument into the output file and is used to add extra SCHAKAL commands, such as BOX.

Mdshak may be useful for more than just visualization purposes as it extracts atomic co-ordinates from the system specification, positions and quaternions. It is written so that only the output routine (called `schakal_out()`) is SCHAKAL specific. This is quite short and can easily be replaced with one tailored to a different purpose.

-b selects binary output in single-precision floating point numbers. Defaults to ASCII formatted numbers.

If any of the compulsory arguments are omitted, you will be prompted to supply a value. In particular you must *always* supply the number of molecules and polyatomic molecules. (This information is not recorded in the dump header and is needed to determine the number of co-ordinates and quaternions in each dump record.) You must specify which pieces of information to extract using the `-c` option.

The dump files must, of course, be part of the same dump sequence; this is carefully checked. They may be supplied as arguments in any order; *dumpext* automatically determines the sequence from the information in the headers. This is not as pointless as it sounds, since the list may be generated using unix shell wildcards which arrange them in alphabetical rather than numeric order.

The output is arranged in columns, one line per time slice. So if, for example you wish to extract positions and quaternions there will be 7 columns corresponding to $x, y, z, q_0, q_1, q_2, q_3$. Multiple components are printed in the integer order of the component, **not** the order specified with `-c`. If binary output is asked for with the `-b` option the order is the same. The numbers are written sequentially as single-precision floating-point numbers (in machine native format) without record delimiters. The records may be easily read in C with an `fread()` call with the appropriate number of bytes or from FORTRAN77 using direct-access unformatted read[1] of the appropriate length records[2] `OPEN(...,ACCESS=DIRECT,LRECL=nnn)`. In the above example there are $3 \times 4 + 4 \times 4 = 28$ bytes in each record.

C.5 Mdshak

Mdshak was written an interface between *Moldy* configurations and a molecular graphics program called SCHAKAL[33]. The output format is sufficiently transparent that it should not be hard to modify it for any other molecular graphics package. Note also that a SCHAKAL file viewer called "read_shak" written by the author for use with AVS, the general purpose visualization program, is available free from the International AVS Center[3]

Mdshak writes a SCHAKAL file containing the co-ordinates of a single or of multiple time slices during a MD run. It can read configurations from (*a*) a system specification file plus lattice start, (*b*) a system specification file plus dump file (*c*) a restart file or (*d*) a restart file plus dump file. It can be driven either interactively or using command-line arguments.

In interactive mode you are prompted for the source of the system specification (sys-spec or restart file) and the configurational information (restart or dump file). But you must either redirect the output (using `>` on unix) or specify an output file with `-o` or the output will go to the screen!

Alternatively, the command line options are

`mdshak [-s` *system-specification* `] | [-r` *restart-file* `] [-d` *dump-file-format* `] [-n` *dump-range* `] [-c] [-i` *extra-text* `] [-o` *output-file* `]`

where the meanings of the options are

-s read a system specification file

-r read a restart file. Only one of `-s` or `-r` may be given.

-c if reading a system specification file, skip any preceding control parameter info.

-d read the configurational information from a dump file. The argument is a dump prototype name containing a `printf()` format string – see section 3.7.

-n range of records in dump file, specified in "selector" format.

[1] The usual FORTRAN sequential unformatted read is not suitable as this expects record size information to be present. *Dumpext* can not write this as its format is entirely determined by the whim of the compiler writer. FORTRAN unformatted sequential files are not guaranteed to be portable even between different compilers on the same machine.

[2] Be aware that the FORTRAN77 standard does not guarantee what units the record length is specified in. Predictably some manufacturers' compilers use bytes and others words. Consult the documentation!

[3] It may be obtained using anonymous ftp from `avs.ncsc.org` or by writing to The International AVS Center, PO Box 12889, 3021 Cornwallis Road, RTP, NC 27709, USA.

section 3.3.2) but is retained in case of machines for which no XDR implementation is available. The command

`dumpconv` *binary-dump-file text-dump-file*

creates a straightforward ASCII text file with a representation of the dump information in the binary file including the header. The command

`dumpconv -d` *text-dump-file binary-dump-file*

converts it back. Seven significant decimals are used which ought to retain almost all of the precision of the single precision binary floating-point numbers. You can also convert an old "native" format dump into XDR format using the `-x` option, *viz*

`dumpconv -x` *native-dump-file xdr-dump-file.*

The `-x` and `-d` options may be combined if the input is a text format dump.

C.4 Dumpext

Dumps are designed so that *Moldy* can take care of all the bookkeeping, perform data security checks and to divide a lot of data into manageable portions. It is therefore not in a convenient form for reading by other programs, especially FORTRAN ones. *Dumpext* is a program which processes dump files, extracts a subset of the information and outputs it in a form more suitable for reading by other programs. It is invoked with the command:

`dumpext -R` *nmols* `-Q` *nquats* `-c` *components* `[-t` *timeslices* `]`
`[-m` *molecules* `] [-o` *output-file* `] [-b]` *dump-file1 dump-file2 ...*

The meanings of its arguments are

-R the total number of molecules. This argument is compulsory.

-Q the number of polyatomic molecules. This argument is compulsory.

-c which pieces ("components") of the information in dump record to extract. This is a selector-format list and may list any combination of

1. C of M positions
2. quaternions
3. unit cell matrix
4. potential energy
5. C of M velocities
6. quaternion derivatives
7. unit cell velocities
8. C of M accelerations
9. quaternion accelerations
10. unit cell accelerations
11. C of M forces
12. torques
13. stress tensor

This argument is compulsory.

-t which timeslices (or dump records) to extract. This is a selector format list and is the index in the whole dump sequence, not just a single file. Timeslices or dump records are sequentially numbered in the dump files from 1. Defaults to all timeslices.

-m extract information for selected molecules. This is a selector list specifying the molecule index. Defaults to all molecules.

-o name of optional output file. Defaults to standard output.

Appendix C

Utility Programs

Five utility programs are included in the *Moldy* package, mostly for manipulation and analysis of dump data (see section 3.7). They are easily compiled on unix systems using the makefile: the command is "`make` *progname*" for each program or "`make utilities`" to make the lot. Apart from *mdshak* which must be linked with several modules from *Moldy* itself they are all self-contained in a single source file. They are written with unix systems in mind using unix-style option arguments, but ought to compile and run under VMS if defined as foreign commands.

Since several of them require you to specify lists of (integer) numbers, for example selected molecules, timeslices *etc* they share a common syntax for these options. For example the command-line argument `-t 1,5,17-20,34-100:10` selects numbers 1, 5, 17 to 20 inclusive and 34,44,54.... Selectors are separated by commas and each one may be a range separated by a hyphen with an optional increment following a colon.

C.1 Moldyext

It is usual to plot the instantaneous values of energy, temperature *etc* during the course of a simulation, for example to monitor the approach to equilibrium. *Moldyext* processes the periodic output produced by *Moldy* (see section 3.5) and extracts this information from each timestep recorded. It is presented in tabular form for input into plotting programs. The command is

`moldyext -f` *fields output-file1 output-file2* ...

where *fields* is a list in the selector format above. The numbering runs from left to right, ignoring newlines and blank columns so that the translational KEs in figure 3.1 are fields 1 and 14 and the σ_{zz} component of the stress tensor is 30.

Moldyext can currently only extract information from the "instantaneous" section of the output, not the rolling averages or standard deviations.

C.2 Dumpanal

Dumpanal examines the dump files given as arguments and prints out the header information to help with identification. For example

`dumpanal` *dump-file1 dump-file2* ...

C.3 Dumpconv

Dumpconv is a tool for moving binary dump files between computers of different architectures. It has mostly been superseded by the portable XDR format dump files introduced in version 2.1 (see

Appendix B

Organization of Moldy

B.1 Data Structures

B.2 Control Flow

```
4          0            0          0 24.31 2 Mg2+
Chloride  8
5          0            0          0 35.45 -1 Cl-
end
MCY
1 1 1088213.2   5.152712        0               0
1 2 1455.427    2.961895        273.5954        2.233264
2 2 666.3373    2.760844        0               0
1 4 47750.0     3.836           546.3           1.253 # New values of Mg potl
2 4 111.0       1.06            0               1.0
1 5 198855.0    3.910           0               0
2 5 1857.0      2.408           77.94           1.369
4 5 28325.5     2.65            0               0
end
```

Appendix A

Example System Specifications

A.1 Argon

```
# LJ Argon - about as simple as you can get
Argon 128
1          0           0          0    36 0 Ar
end
Lennard-Jones
1 1 3984.08 3.41
```

A.2 TIPS2 Water

This is the four-site water model of Jorgenson[30]. Only the oxygen site interacts via the Lennard-Jones potential, and the charge site, M, is displaced 0.15Åfrom the Oxygen.

```
# Modified TIPS2 water
Water 64
1          0            0          0 16    0 O
2  0.7569503            0 -0.5858822 1 0.535 H
2 -0.7569503            0 -0.5858822
3          0            0      -0.15 0 -1.07 M
end
lennard-jones
1 1 0.51799  3.2407
end
```

A.3 Aqueous $MgCl_2$ Solution

This is a three-component system consisting of MCY water[24], Magnesium and Chloride ions. The Mg^{2+} potential was fitted to the SCF calculations of Dietz and Heinzinger[31] and the Cl^- to the calculations of Kistenmacher, Popkie and Clementi[32]. Note that the potential parameters are expressed in kcal mol^{-1}, and the control file must set the parameter `time-unit=4.8888213e-14`.

```
# MCY Water/ Mg2+ / Cl - solution
Water 200
1          0            0          0 16    0 O
2  0.7569503            0 -0.5858822 1     0.717484 H
2 -0.7569503            0 -0.5858822
3          0            0    -0.2677 0 -1.434968 M
Magnesium 4
```

is transformed into

```
for(processor = 0; processor < NPROCESSORS; processor++)
   for(i = processor; i < N; i += NPROCESSORS)
      <body of loop>
```

so that the outer loop is over the number of available processors, and the inner loop has an increment of that number. Note that this transformation reorders the loop iterations.

If the outer loop is to execute concurrently, each sub-process or thread must have local copies of any temporary variables, notably `i`. Therefore the contents of each loop are isolated into a separate function. Each thread executes a separate incarnation of the function. A shared-memory model is assumed, in which each thread accesses a single copy of the parameters and global variables, but has a unique copy of all of its local variables.

The "master" thread maintains separate copies of result variables and arrays, (*i.e.* the forces, potential energy and stress virial) so that each incarnation of the function can write to its own copy without interference from other threads. After all of the threads recombine, the separate copies are summed to give the final result of the calculation. This method avoids the need for explicit synchronization between threads, but will add to the serial overhead on highly parallel machines.

The method of actually parallelizing the outer loop will vary from machine to machine. The stellar compiler is able to generate parallel code, and the directive `/*$dir no_recurrence*/` instructs it to parallelize the following loop. On other machines this may not be so easy!

This mechanism only works for vectorizing compilers which have a directive which means "ignore dependencies in the following loop". Some, notably the Alliant FX/C compiler take the alternative approach of having directives to the effect of "ignore dependencies involving the following variables". In that case the directives must be inserted "by hand", though even then the `VECTORIZE` directive al least identifies the appropriate loops.

Unfortunately this method had been obsoleted by the ANSI C standard, which makes it impossible to insert directives using the preprocessor. With more modern compilers such as Cray scc the same effect may be achieved using "ignore dependencies" command-line directives.

4.3 Modifying Moldy

4.3.1 Adding a New Potential

By default *Moldy* supports potential functions of the Lennard-Jones, six-exp and MCY forms. However it should be very easy to add further types. The program is written in a highly modular fashion so that *the only code which need be altered is in file "kernel.c"* (and occasionally in "defs.h").

The calculation of the potential and forces is performed entirely in the function `kernel()`. This function is called repeatedly with a vector of (squared) distances between some reference site and its neighbour sites. Vectors of potential parameters and charges are supplied which bear a one to one correspondence with the elements of the distance vector. It calculates the corresponding values of $\frac{dU(r_{ij})}{dr_{ij}}$ which it stores in `forceij[]`. There are several variants of the force-calculation loop, one for each kind of potential. The potential type in use is passed as a parameter to `kernel()` and is used in a `switch` statement to select the appropriate code.

To add a new potential the array of structs called `potspec[]` must be extended. The new array element should contain the name of the new potential (against which, the names given in system specification files will be matched) and the number of potential parameters for each site pair[3]. Then define a new preprocessor symbol to the index of the new type in the array `potspec[]` (after the line `#define MCYPOT 2`. The value must correspond to the index of the new entry in `potspec[]` (starting from 0 in accordance with the usual C convention). This constant should be used to define a new case in the `switch` statement of `kernel()`, and this is where the code to evaluate the potential goes.

The existing cases may be used as a model, especially for the evaluation of the electrostatic term $\mathrm{erfc}(\alpha r)/r$ which is evaluated by the polynomial expansion of Abramowitz and Stegun[28, section 7.1.26]. There are currently *two* versions of each loop, the second omitting this term for efficiency when all the electric charges are zero (which case is flagged by a negative value of α).

Finally, the distant potential correction for the new potential should be added as a new case to function `dist_pot()`. The code should evaluate

$$\int_{r_c}^{\infty} r^2 U(r) dr$$

for the potential $U(r)$.

4.3.2 Parallel Version

An experimental parallel version of *Moldy* is under development, and currently runs on a Stellar GS1000. It consists of replacements for files "force.c" and "ewald.c".

The approach taken is to retain the vectorized inner loops and parallelize the the outer loops, over cells for "force.c" and **k** vectors for "ewald.c". The two parallel loops are structured by the method developed by the Kingston group[29]. That is, a loop of the form

```
for(i = 0; i < N; i++)
   <body of loop>
```

[3] By default up to five parameters are allowed. If this is not sufficient, the limit is set by the value of the constant `NPOTP` defined in "defs.h"

and optimizations are required for ultimate performance, and inevitably portability must be sacrificed to gain speed.

Nonetheless, it has been found that following a few basic rules gives extremely good performance on a range of vector computers.

- Write vector loops simply to make minimum demands on the compiler's ability to vectorize.
- Minimize the floating-point operation count.
- Minimize memory references in vector loops.
- Ensure that arrays are accessed with a stride of 1. (This makes a large difference on some machines, notably the Stellar, and little on others).
- Minimize integer arithmetic in critical code. The CRAY in particular has no integer multiplication hardware, which operations are slow as a result.
- Make use of manufacturer's linear algebra library functions.

The performance of *Moldy* has been carefully studied using profiling tools, and all critical regions of code are written as efficiently vectorizable loops.

The most critical sections of code (*i.e.* those which use the majority of the computer time) are all to do with the site forces calculation. Thus it is the inner loops in "force.c", "ewald.c" and "kernel.c" to which most attention should be paid. The pair-distance loop of `rdf_calc()` in "rdf.c" should vectorize for efficient radial distribution function evaluation. Others which are of minor importance are in "beeman.c", "matrix.c", "quaterns.c" and "algorith.c". "Auxil.c" contains alternative versions of various sum, dot product, scatter and gather routines *etc* which are interfaces to machine-specific libraries *e.g.* Cray scilib, Convex veclib (which usually have FORTRAN calling conventions). There are also default versions coded in C which do vectorize, for machines lacking specialist libraries as well as for scalar computers.

The program should, of course, be compiled with options specifying vectorization. Many compilers also have an option which directs it to ignore recurrences throughout the whole program, *e.g.* `-va` on the Convex, `-va` and `+CaliasFreePointers` for the Stellar and `-h ivdep` on the Cray compilers. It should normally be safe to use these options. Since highly optimizing and vectorizing compilers frequently contain bugs, and since some options generate "unsafe" optimizations, it may be necessary to restrict the highest optimization level to those modules which contain critical code.

To allow the compiler to generate vector code, machine-specific compiler directives must be inserted into the source. The reason is that the run-time dimensioned arrays necessary to implement such a flexible program must use pointers as their base. (See any C textbook, *e.g.* Kernighan and Ritchie[27] for an explanation of C pointers and arrays.) Unfortunately this means that the compiler can not determine that each iteration of the loop is independent of the preceding iterations. In the jargon of vectorizing compilers, there may be a *vector dependency* or *recurrence*. Each manufacturer's compiler has its own peculiar method to inform it that despite appearances there is no dependency. For example the CRAY compilers use a `#pragma ivdep` statement whereas the convex and Stellar compilers use a "significant comment" `/*$dir no_recurrence*/`.

A mechanism is provided to insert appropriate directives using the C preprocessor. The text `VECTORIZE` has been placed before each loop which ought to be vectorize, and the file "defs.h" contains machine-conditional `#define`s to replace it with the appropriate directive. Currently directives for the CRAY, Stellar and Convex compilers are included, and null text is substituted for other machines. Notice that in each case the substituted text is *not* the directive described in the manual, but rather that directive *after* it has been passed through the preprocessor. To determine what should be substituted on a new vector machine, create a small test containing the documented directive and use the C preprocessor on that file. The output will show the form that should be defined in "defs.h".[2]

[2] Note that there is a bug in the Ardent (now Stardent) compiler whereby the `#pragma ivdep` directive acts syntactically like a C statement. There are several loops in "force.c" and "ewald.c" where the body of an `if` clause is a single `for` loop without braces. The directive forces the `for` loop *outside* the scope of the `if` clause, erroneously altering the program flow. The solution is to modify the code by putting braces around the directive and the `for` loop.

4.2 Portability

A major goal in writing *Moldy* was that it be as portable as possible between different computers and operating systems. It is written in the Kernighan and Ritchie[27] compatible subset of ANSI C and assumes the library calls and header files defined for a hosted implementation of the standard. It should therefore be possible to compile and run *Moldy* on any computer which has a good C compiler.

There are two sources of difficulty in moving *Moldy* to a new machine. The library may not contain some necessary ANSI routine, or may not have the required header files. Secondly, to make good use of vector or parallel architectures, compiler directives or calls to specialized library functions are usually required. Replacement ANSI library functions are supplied for the VAX/VMS, unix (both Berkeley and AT&T system V varieties) and COS operating systems. Different versions of a function are selected by the C preprocessor and conditionally compiled according to the pre-defined preprocessor symbols (see the documentation for your compiler). For ease of portability *all system-dependent functions are in the module "auxil.c"* and *all preprocessor conditionals are in the header file "defs.h"*.

If the target machine has ANSI conformant C libraries, all that must be done is to define the preprocessor symbol `ANSI_LIBS`, either in "defs.h" or by using a compiler option *e.g.* `-DANSI_LIBS`. This is done automatically in "defs.h" for several machines known to have conformant environments. If the target operating system is the system V variant of UNIX, the preprocessor symbol `USG` is defined automatically in "defs.h". It is possible that some environments may defeat the selection making it necessary to define it by hand, either in "defs.h" or by setting the compiler option `-DUSG` in the makefile. In addition, if the symbol `HAVE_VPRINTF` is defined, it is assumed that the library contains the function `vprintf()`, otherwise the replacement in "auxil.c" is used. If your system does not have `vprintf()` but does have `_doprnt()` then define the symbol `HAVE_DOPRNT` to use it.

4.2.1 System Dependencies

In this section, details of system-dependent functions are described for the major operating systems.

- The ANSI header file "string.h" is missing from Berkeley unix, and incomplete in other systems, as is "time.h". Other files such as "stdlib.h" and "stddef.h" are new with ANSI C. Replacements are included which may be dispensed with on an ANSI conformant system - If the symbol `ANSI_LIBS` is defined they simply include the system version.

- The supplied `clock()` function on UNIX systems resets to zero after 36 minutes. Replacements, called `cpu()` for both system V and Berkeley UNIXes are supplied in "auxil.c". (Otherwise `cpu()` simply calls `clock()`.)

- The ANSI function to delete a file, `remove()`, the signalling function `raise()` and the string functions `strstr()` and `strerror()` are missing from pre-ANSI libraries. Replacements are supplied in "auxil.c".

- Replacements are provided for functions `memset()`, `memcpy()` and `strchr()` which are missing from Berkeley UNIX.

- The function `vprintf()` is often absent from older libraries. Replacements are provided which *a)* call the internal function `_doprnt()` (or `XPRINTF()` in the case of COS) or *b)* implements a portable `vprintf()`. Use the preprocessor macros `HAVE_VPRINTF` or `HAVE_DOPRNT` to select which.

- "Auxil.c" contains the functions `replace()` and `purge()`. `replace()` renames a file, making a backup of any existing file of that name. `purge()` removes the previous or backup version of a file. The pure ANSI versions just interface to `rename()` and do nothing respectively.

4.2.2 Optimization and Vectorization

Moldy has been designed to run fast on a wide range of computers, and in particular on those with vector architectures. This is a difficult problem, since necessarily machine-specific compiler directives

Chapter 4

Compiling and Modifying Moldy

The *Moldy* distribution consists of numerous files of 'C' source code for *Moldy* and the utility programs, command or job files to compile the source, LaTeXinput for the manual and example control and system specification files. For ease of transport these are packed into one large "archive" file, whose format and method of unpacking depends on the operating system of the target machine. At present it is available for:

unix The archive is either a tar archive called *moldy.tar* or a Bourne shell script called *moldy.shar*. The first is unpacked by the command `tar xvf moldy.tar` and the second by `/bin/sh moldy.shar`.

VMS The archive is a DCL command file called *moldy.com*, and is unpacked by the command `@moldy`.

MS-DOS The files must be unpacked from the tar archive on another host and transferred to the PC by disk or ftp.

COS The archive is a JCL job file. It does not contain the manual, examples or utilities, which are only of much use on the front-end system. N.B. For unicos, use the unix distribution.

4.1 Compilation

The source code of *Moldy* consists of 22 files of C programs (which have suffix ".c") and 8 header files (suffix ".h"). To build *Moldy* , all of the ".c" files must be compiled and linked together to form an executable. The method of doing this depends on the operating system of the target computer.

unix The "make" program is used and the make file is supplied in the distribution. You will have to edit this file, called "Makefile" to select the set of compiler flags appropriate for your machine (see the next section for what to do if it is not catered for). Then just type `make` to compile and link *Moldy* and `make utilities` for the utility programs.

VMS Simply type `@compile` to execute the command file "compile.com". This will build *Moldy* and the utilities.

DOS There is a makefile for Borland Turbo C called Makefile.mak in the standard distribution. This must be edited to select the appropriate compiler options before executing "make"[1] Alternatively the programs may be built within the interactive environment. Consult the makefile to find out which source files to like to build *Moldy* and the utilities.

COS In contrast to the other cases the cray COS distribution is a self-contained JCL file which, when executed, will compile and link *Moldy*.

[1] Be sure to delete or rename the unix make file "Makefile" since Turbo C "make" will attempt to execute this in preference to "Makefile.mak"

fill space to construct the required infinite framework. This is notified to the program by modifying the first line of the specification of that molecule to read

species-name$_i$ `1 Framework`
⋮ ⋮ ⋮ ⋮ ⋮ ⋮ ⋮

(compare with section 3.2.1). The effect of the special keyword `framework` is

1. to remove the rotational freedom of the molecule. This preserves the infinite structure over MD cell repeats by disallowing relative motion of its parts. (Linear motion does not destroy the structure and *is* allowed.)

2. to modify the effect of the periodic boundary conditions. Normally a molecule is assumed to be "small" and periodic relocations are applied to *all* of its atoms depending on its centre-of-mass co-ordinates relative to some interacting molecule. In contrast, the atoms of a framework are independently relocated. This ensures that each molecule "sees" all framework atoms from any unit cell which are within the cut-off distance.

In the present version of the program, only one framework molecule is allowed, though more may be permitted in future versions. Consequently the configuration given as a lattice start must fill the entire MD box. (A skew start is not sensible under these circumstances since the orientation of the framework must be explicitly specified to construct a good space-filling structure.)

3.11 Messages and Errors

Apart from the periodic output, there are occasional "once-off" messages which *Moldy* writes to the usual output file. Such messages begin with the characters `*I*`, `*W*`, `*E*` or `*F*` denoting the classes *information*, *warning*, *error* or *fatal* respectively. Their meanings are

`*I*` information. These are often produced by subroutines to give useful information on their particular calculations. For example when temperature scaling is turned off a message to that effect is recorded in the output file. Various routines which calculate internal quantities such as the Ewald sum self energies and distant potential corrections also record their values using an information message.

`*W*` warning. When the system specification is suspicious but not clearly wrong, or some untoward condition is detected such as two atoms approaching too closely, a warning message is issued.

`*E*` error. Error messages are issued when an mistake is detected reading any of the input files. To make correction easier, processing continues until the end of that file, so that all of the errors are found. The simulation is then stopped with a fatal error.

`*F*` fatal. The simulation is terminated immediately. Faulty input files generate fatal errors after they have been completely processed. There are many other conditions which also generate fatal errors, for example if the simulation algorithms violate some criterion such as quaternion normalization or constraints (see section 2.1), if the program runs out of memory or if a restart file can not be correctly opened or is of the wrong format.

The text of the messages is intended to be self-explanatory.

Thus the default constraint of $\boldsymbol{h}_{?1} = \boldsymbol{a} = (a_x, 0, 0), \boldsymbol{h}_{?2} = \boldsymbol{b} = (b_x, b_y, 0)$ is given by `strain-mask=200` (8+64+128). Another useful value is 238 which freezes all the off-diagonal components. This is needed for a liquid simulation since there are no shear restoring forces acting on those components.

3.9 Cutoffs and Adjustable Parameters

There are four parameters related to the Ewald sum method of force evaluation (see section 2.4), α, r_c, k_c, and `subcell`. The optimum choices for these are discussed in an excellent article by David Fincham[26]. In addition the two options `strict-cutoff` and `surface-dipole` select how rigorously the real-space cutoff is applied and whether to include the De Leuuw durface dipole term.

The α parameter is specified by `alpha` and the direct and reciprocal space cutoff distances r_c and k_c by `cutoff` and `k-cutoff`. They are specified in units of Å^{-1}, Å and Å^{-1} respectively. Before a new simulation is started, trial runs should be performed to establish suitable values of these three parameters. Two variants of the cutoff algorithm are allowed; use the parameter `strict-cutoff` to choose which. For a simulation of 512 MCY water molecules the values $\alpha = 0.3\text{Å}^{-1}$, $r_c = 9\text{Å}$ and $k_c = 1.9\text{Å}^{-1}$ give potential energies correct to approximately 1 part in 10^5. For a simulation including ions - 1.1 Molal Magnesium Chloride solution - the same accuracy is attained with $\alpha = 0.45\text{Å}^{-1}$, $r_c = 9\text{Å}$ and $k_c = 3\text{Å}^{-1}$.

The other relevant parameter is the switch `surface-dipole` which includes the dipole surface energy term of De Leeuw, Perram and Smith[18]. See the note in section 2.4 for an explanation of why this term should *never* be used for an ionic (as opposed to dipolar) system.

The two adjustable parameters which control the link-cell force calculation (see section 2.5) are `subcell` and `strict-cutoff`. The former specifies the length (in Å) of the side of a link cell and determines the number of cells the MD cell is divided into. In fact the MD cell is divided into a whole number of subcells whose side in each of the three directions is nearest to the value of `subcell`. (The default of zero though, is special and sets subcell to one fifth of the cutoff radius.) In general the smaller the link cell, the more accurately the cutoff radius is implemented, but too many of them reduces the efficiency of the program.

In the default cutoff mode `strict-cutoff=false` the list of neighbour cells is constructed to include all cells whose centre-centre distance is less than the cutoff. This means that some molecule pairs separated by more than the cutoff will be included and some by less will be omitted. Setting `strict-cutoff` to true generates a larger cell neighbour list which is guaranteed to include all appropriate molecule pairs. Furthermore, molecules separated by more than the cutoff are excluded from the force calculation by setting their separation to a large number, 100 times the cutoff, at which distance it is assumed the potential is very close to zero. This is therefore the mode of choice for liquid simulations where any artificial anisotropy is undesirable. See section 2.5 for a full explanation.

It is worth noting that it is unnecessary to recompile the program or change anything else when the cutoffs are modified. Unlike most MD codes, *Moldy* employs dynamic array allocation and automatically sets up arrays of the correct size (and no more!) for any given k_c. In the case of r_c any value up to the length of the smallest MD cell edge may be given. Any attempt to exceed this will result in the program terminating with an error[14].

3.10 Framework Simulations

There has recently been much interest in simulations of systems of molecules interacting with some rigid framework such as zeolites, clays and other surfaces. *Moldy* has the capability to include such a framework in a simulation by defining it as a special kind of molecule.

The system specification should contain an entry, similar to that for a normal molecule, which describes the atomic sites belonging to one MD cell's worth of the framework. Its periodic images should

[14] In the unlikely event this limit is too small, the modification is simple. Simply set the `#defined` constant `NSH` in `force.c` to the maximum number of MD cell lengths required and recompile. This will use slightly more memory, but not unreasonably more

a timestamp), common to all the files in a dump run, and the timestamp[13] of any restart file used to start the run.

It is not possible to dump directly to magnetic tape. *Moldy* must rewind to the beginning of a file to keep the header up to date with the number of dumps in the file, as well as extend existing files. Neither operation is allowed on a tape drive. Large disk stores are now very cheap so this should not be a problem in practice. If disk store *is* limited then the simulation may be divided into multiple *Moldy* runs interspersed with copying of dump files to tape.

Notice that *Moldy* must sometimes read an existing but complete dump file to propagate the unique marker to all of the files in a sequence. Therefore when continuing a simulation and a dump run, at least the immediately preceding dump file must still be accessible. This should be borne in mind when copying dumps to tape!

Moldy is careful to ensure that existing files are not overwritten - especially necessary since dump records are added to the end of an existing dump file. Whenever *Moldy* prepares to start a new dump file it checks to see if one of that name is already present. If so, a new name is chosen by "mutating" the old one, and a warning message to that effect is written to the output file. On the other hand, if the *first* file of a new dump run (including one initiated because of some error in continuing an old one) already exists, the *prototype* file name is mutated as above and the whole dump run is written to files based on the mutated name.

When a run is restarted checks are made to ensure that the values of the dump control parameters have not been altered. If they have, it is not possible to continue an existing dump sequence and a new one will be started. (If existing dump files are present the new sequence will have mutated file names.) This also happens if an existing file does not appear to be a *Moldy* dump. Existing dump files are also tested to ensure that there is no corruption (due, for example to a system crash) and that they contain the correct number of records. If the dump sequence can not be continued in these circumstances, *Moldy* terminates with a fatal error rather than waste computer time.

Two utility programs included in the distribution are *dumpanal* which identifies dump files by printing out the headers and *dumpext* which extracts atomic or molecular trajectories. The latter should be useful as a prototype for writing programs to analyse dump data.

It is frequently convenient to perform analysis of dump data, and perhaps graphical output on a different computer to that which generated the data. In the past it has not usually been possible to sensibly transfer binary data between computers of different architectures. However *Moldy* is able to write dump files in a portable format using XDR (see section 3.3.2) which may be read by *dumpext* on any machine. The control parameter `xdr` enables XDR mode for dumps as well as restart files. As yet, XDR is not available on every machine. Therefore a program called *dumpconv* is provided which converts dump files to a portable text file format (which may be easily moved between machines) and back again. It is described in appendix C.3.

3.8 Constant Stress Simulation

Setting the control parameter `const-pressure` switches from a constant-volume simulation to a constant-stress simulation using the method of Parrinello and Rahman[22] (see section 2.7). The value of the "MD cell" mass parameter, W is given by the control parameter `w` and the external pressure by `pressure`. At present it is not possible to specify an anisotropic external stress, though this capability may be added in future versions of the program.

The $\boldsymbol{h}$ matrix may be constrained so as to disable motion of any or all of its components using the parameter `strain-mask`. `Strain-mask` is a bitmap: each "bit" of the integer freezes one of the components of $\boldsymbol{h}$; bit i freezes $\boldsymbol{h}_{kl}$ with $i = 3(k-1)+l-1$. The bitmask is the sum of 2^i over the i's to be set, so the `strain-mask` values

$$\begin{pmatrix} 1 & 2 & 4 \\ 8 & 16 & 32 \\ 64 & 128 & 256 \end{pmatrix} \text{ constrain the corresponding components of } h, \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}.$$

[13] A timestamp is simply the number of seconds elapsed since midnight on January 1, 1970.

3.7 Dumping

The dump facility is provided in order to allow the calculation of dynamic properties, such as time correlation functions and additional static averages not normally calculated by *Moldy*. During a run, dump files are produced which contain a record of the simulation dynamic variables (positions, quaternions *etc*) at varying degrees of detail. Any property of interest, dynamic or static, may then be evaluated using the data in the dump.

A dump consists of a sequence of files since the amount of data generated in a run can be very large indeed and it is usually more convenient to manipulate a series of smaller files rather than one large and unwieldy one. *Moldy* takes considerable pains to ensure that a contiguous sequence of dump files is maintained and also ensures that dumps from different runs are not accidentally intermixed. There is no requirement that a dump file be produced by a single run of *Moldy* , which extends an existing file or starts a new one as appropriate. A simulation may stop and be restarted many times without disturbing the dump file sequence. The sequence should (in most cases) even survive a system crash and a restart from a backup file (see section 3.3.1).

Each dump file in a sequence is a binary file consisting of a *dump header*, which contains information about the contents of the file followed by a number of *dump records* which contain the actual data.

Several control parameters govern dumping. It starts at the timestep specified by `begin-dump`, and a dump record is written every `dump-interval` timesteps thereafter. After `ndumps` dump records have been written to a file, it is closed and another is begun. Filenames are generated from a prototype (given by the parameter `dump-file`) by appending a number, so that if the prototype is *MDDUMP* then successive files will be named *MDDUMP0*, *MDDUMP1*, *MDDUMP2 etc* If it is not convenient for the sequence number to appear at the end of the file, include the characters "%d" at an appropriate point[12]. For example under VMS, specifying `dump-file=mddump%d.dat` will name the files *mddump0.dat*, *mddump1.dat etc*.

Each dump record is a sequence of single-precision floating point binary numbers. These are written either in native (*i.e.* the machine's own) format or XDR format (see section 3.3.2) depending on the value of the control parameter `xdr`. The record's exact contents are determined by the control parameter `dump-level` which is a bit flag *i.e.* a value of 2^n means that bit n is set. Four bits are used and any combination may be specified but the cumulative values 1, 3, 7 and 15 are most useful. A value of 0 disables dumping. The data dumped for each bit is as follows:

bit 0 centre of mass co-ordinates, quaternions, unit cell matrix and potential energy.

bit 1 centre of mass velocities, quaternion and unit cell matrix derivatives.

bit 2 centre of mass accelerations, quaternion and unit cell matrix second derivatives.

bit 3 forces, torques and stress tensor.

Items selected are written in the order laid out above. Within each set of variables, values are ordered primarily by species in the order they appeared in the system specification. Within a species ordering vis by molecule (or atom) and at the finest level by x, y or z component ($q_0, \ldots q_3$ for quaternions). Therefore if n is the total number of molecules and n_r is the the number with rotational freedom the size of each record is

$$\begin{array}{rll} & 3n + 4n_r + 9 + 1 & \text{(if bit 0 is set)} \\ + & 3n + 4n_r + 9 & \text{(if bit 1 is set)} \\ + & 3n + 4n_r + 9 & \text{(if bit 2 is set)} \\ + & 3n + 3n_r + 9 & \text{(if bit 3 is set)} \end{array}$$

single precision floating point numbers.

The header is a copy of a `struct dump_t` (see appendix B.1 for the format). It contains the simulation title and version number, the timestep at the beginning of the file, the control parameters `dump-interval` and `dump-level`, the maximum and actual number of dump records in the file, a unique marker (actually

[12] This is actually the code "sprintf()", the C library function uses to signify converting an integer to a decimal character string. This function is used to create the actual file name from the prototype and the integer dump sequence number. (See any C library manual for details.)

Figure 3.2: Example output of radial distribution functions. After the header line consisting of underscores there is an indication of the bin width b (that is the distance between points at which the RDF is tabulated). Then for each site type pair $\alpha\beta$ there is a line listing which pair (*e.g.* `O-O RDF`) followed by `nbins`. values of $g_{\alpha\beta}((i+1/2)b)$.

```
------------------------------------------------------------------------------
     Radial Distribution Functions   Bin width=0.1
     O-O RDF
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000481 0.035710 0.183334 0.442186 0.613992 1.024402
1.046396 0.964906 0.830174 0.660035 0.693341 0.615902 0.593192 0.510595 0.530697 0.532030 0.535959 0.524457 0.523221 0.466219
0.496028 0.438487 0.456500 0.410547 0.443861 0.457956 0.446822 0.452202 0.419768 0.439333 0.465509 0.486887 0.461970 0.475745
0.478883 0.480854 0.509090 0.533728 0.552747 0.552555 0.575402 0.547278 0.544836 0.493597 0.488168 0.520727 0.508073 0.479948
0.501159 0.484000 0.485378 0.489160 0.464448 0.466791 0.476508 0.446576 0.470948 0.474468 0.449340 0.462169 0.501220 0.519107
0.513338 0.510192 0.499766 0.525963 0.504663 0.517673 0.498359 0.512156 0.507061 0.466390 0.464342 0.445886 0.417555 0.407778
0.387220 0.374041
     O-H RDF
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 26.976688 0.000000 0.000000 0.000000 0.000000
0.000000 0.016214 0.061257 0.304082 0.647342 0.847404 0.757188 0.601222 0.478273 0.462682 0.449614 0.450424 0.518998 0.572242
0.689704 0.914269 1.184674 1.441772 1.570390 1.609068 1.600392 1.430457 1.322722 1.183606 1.103701 1.061788 0.980018 0.960570
0.924390 0.908883 0.877591 0.857668 0.890761 0.852463 0.815447 0.824963 0.841255 0.890416 0.929030 0.960589 0.984145 1.020650
1.028199 1.047496 1.064600 1.099812 1.095715 1.073793 1.078131 1.049212 1.052160 1.052001 1.020737 1.010782 0.979748 0.983158
0.988946 0.967620 0.955655 0.944384 0.952145 0.948509 0.946692 0.960097 0.959299 0.964074 0.969219 0.972704 0.998504 1.027791
1.041576 1.037637 1.039961 1.016804 1.004726 1.026805 1.030903 1.006268 0.972421 0.948140 0.908959 0.877089 0.849855 0.817964
0.776986 0.721485
```

mean square forces) the components are printed across the page[10]. In addition to those variables printed as part of the periodic output, the pressure, the virial, mean square forces, mean square torques and total dipole moments are calculated.

3.6 Radial Distribution Functions

Radial distribution functions are calculated by binning site pair distances periodically throughout the simulation (see section 2.8. As this process is expensive in computer time the binning subroutine is invoked only every few timesteps, as set by the control parameter `rdf-interval` (20 by default). Since the pair distances only change a little on each timestep, very little statistical information is lost. Collection of binning data may also be turned off during an equilibration period: specify when binning is to start by means of the parameter `begin-rdf`. The parameters `rdf-limit` and `nbins` control the details of binning, giving respectively the largest distance counted and the number of bins that interval is divided into[11].

Every `rdf-out` timesteps (by default 5000) the RDFs are calculated from the binned distances and printed out, and the counters are reset to zero to begin accumulation again. Distances are binned and RDFs $g_{\alpha\beta}(r)$ calculated separately for each distinct type of atom-atom (or site-site) pair. An explanation of the output format is given in figure 3.2. Note that each number should be considered as the value at the *centre* of its bin, so that entry i in each list is the value of $g_{\alpha\beta}((i+1/2)b)$ where b is the bin width.

There are a couple of tricks which may be played with the system specification if the atomic pair RDFs do not give exactly the functions required. Firstly, It is possible to calculate rdfs about a particular site, distinguishing it from otherwise identical atoms by assigning it a new and unique site id in the system specification file. (This is the MD equivalent of the isotopic substitution method used in neutron diffraction). Secondly, if the *molecular* pair correlation is required, this is identical to the RDF of an atom located at the molecular centre-of-mass. A "ghost" site without charge, mass or potentials may be added if necessary.

[10] Remember that the standard deviation is a measure of the *fluctuations* about the mean, **not** the *uncertainty* in the mean. For that the standard error in the mean is required, which is more difficult to evaluate. Theoretically it is the *s.d.* divided by $\sqrt{N}$ where N is the number of independent observations. But successive timesteps are highly correlated and do not count as independent.

[11] The calculation of the interatomic distances is done on the basis of the "minimum image" convention. Consequently the calculated value of $g_{\alpha\beta}(r)$ will not include all possible pairs if $r_c > L/2$ and $g_{\alpha\beta}(r)$ will tail off to zero. (Here, the cutoff r_c is the one specified by `rdf-limit`.) This is in contrast to the more sophisticated behaviour of the force calculation.

Figure 3.1: Sample *Moldy* output from a simulation of a two component mixture. The first component is a polyatomic molecule and the second is atomic. There are three *frames*, for the instantaneous values, the rolling averages and their associated standard deviations. Within a frame, each row has the following meaning: for translational and rotational kinetic energies and temperatures it is the per-species value; for the potential energy it is the direct and reciprocal space components, and the MD cell matrix, `h` and the stress are laid out as 3×3 matrices.

```
          Nov 17 15:13:06 1989          Water_test          Page 4
Trans KE    Rot KE    Pot Energy    Tot Energy    TTemp    RTemp    Temp    h(1,*)   h(2,*)   h(3,*)       Stress   Stress   Stress
======== Timestep 10 Current values ==================================================================
   243.88    453.88     -187.35         533.5     305.5     568.6    424.4     12.53     0.00     0.00          589     46.4      120
   22.053         0      1.0401                   221.0       0.0               0.00    12.53     0.00         46.4      373     90.1
                                                                                0.00     0.00    12.53          120     90.1    -207
---- Rolling averages over last 10 timesteps ---------------------------------------------------------------------------
   240.27    319.31     -82.472        533.39     301.0     400.0    342.9     12.53     0.00     0.00     1.2e+03      296      127
   22.077         0      34.205                   221.3       0.0               0.00    12.53     0.00          296      589      133
                                                                                0.00     0.00    12.53          127      133     -132
---- Standard deviations ------------------------------------------------------------------------------------------------
   1.8214    71.893      56.441        .19942       2.3      90.1     43.4     0.00     0.00     0.00     1.32e+03      750       51
    0.013         0      17.173                     0.1       0.0               0.00     0.00     0.00          750      119     55.2
                                                                                0.00     0.00     0.00           51     55.2       49
```

The options for scaling separately rotation and translation, and per species may be useful for achieving equilibration in "difficult" systems where mode-coupling is ineffective. In those situations it is otherwise possible for all the energy to be transferred into the rotational modes of a particular species, halting any progress to equilibrium for other degrees of freedom. These options ensure that all degrees of freedom have some thermal energy.

The option controlled by bit 3, to discard all existing information and start from a random set of velocities may be of use when starting from far-from-equilibrium situations. In such cases the forces are frequently so large that the velocities and accelerations exceed the limits of the integration algorithm and timestep, which results in *Moldy* stopping with a *quaternion normalization* or *quaternion constraint* error. Judicious use of this option every few timesteps (using `scale-interval`) ought to allow the system to relax to a state sufficiently close to equilibrium for normal scaling to take over.

Bit 2 is intended to deal with the problem of setting the temperature accurately using scaling. The *ensemble average* kinetic energy which characterizes the temperature of the system and the instantaneous value fluctuates about this value. However in the traditional implementation of scaling, velocities are multiplied by a factor of $\sqrt{desired\ KE/instantaneous\ KE}$. Thus the scaling factor is "wrong" by the ratio of the instantaneous to average KE's which means that the temperature can not be set more accurately than the relative size of the fluctuations in the KE.

The option selected by bit 2 goes some way towards the ideal scaling factor by using the rolling average KE instead of the instantaneous value. The fluctuations in this short-term average should be much lower than in the instantaneous value, allowing more accurate temperature control.

3.5 Output

At the beginning of each run *Moldy* writes a *banner page* containing a summary of the system being simulated and details of the important control parameters. The bulk of the output file is the *periodic output* which contains the instantaneous values of various thermodynamic variables, their rolling averages and associated standard deviations. The *rolling average* is just the mean over the preceding n timesteps where n is set by the control parameter `roll-interval`. An annotated example is given in figure 3.1. The frequency of periodic output may be altered by setting the control parameter `print-interval` to the interval required. (This may be necessary to constrain the size of the output file which can grow to be very large indeed with the default interval of only 10.)

As well as the "short term" rolling averages, long term averages are calculated and printed out at regular but usually infrequent intervals. Accumulation starts on the timestep given by the control parameter `begin-average` and every `average-interval` timesteps thereafter, the means and standard deviations are calculated and printed. This output is interspersed with the periodic output and is formatted with one variable to a line in the form *mean +/- s.d.*. Where a variable has more than one component (such as multiple species for the translational temperature or cartesian components for the

are taken from the backup file and the control file (and a restart file if one is specified) is ignored[6]. In consequence, if a run is aborted or stops abnormally for some reason, the backup file must be removed "by hand" otherwise next time a run starts, the unwanted simulation will continue instead.

If a run terminates abnormally there may also be a *lock file* called `MDBACKUP.lck` which ought to be removed. *Moldy* attempts to prevent two runs from overwriting each other's backup files by creating a lock file whose name is formed from the backup name by appending `.lck`. A second run which attempts to use the same backup file will test for the presence of the lock file and abort the run if it finds it.

A restart or backup file is created by first writing the data to a temporary file which is then renamed to the final name. This ensures that there is no possibility of a file being left incomplete or corrupt if the computer crashes part-way through the write. If the file already exists either it is replaced (on systems which only keep one version of a file) or a new version is created (on systems such as COS or VMS which retain multiple versions). In the unlikely event of it being necessary to change where the temporary file is kept[7], it may be specified with the control parameter `temp-file`.

3.3.2 Portable Restart Files

Moldy is able[8] to read and write restart and dump files in a portable binary format which is transportable between computers of different architectures. So a restart file written on, for example, a Sun may be used to initiate a new run on a Cray, and the dump files generated on the Cray may be analyzed on the Sun. This feature will also be of considerable use in modern heterogeneous networks where diverse machines frequently share a common file space.

The format is based on Sun Microsystems XDR protocol[25]. The XDR routines are available on almost every modern Unix machine, and are simple enough to implement on any other system[9]. If the control parameter `xdr` is set to 1 then all files will be written using this format. *Moldy* automatically determines whether a restart file was written using XDR (by examining the file header) and reads it in the appropriate fashion irrespective of the value of `xdr`.

3.4 Setting the Temperature

To control the temperature of the simulated system, *Moldy* implements the velocity rescaling technique described in section 2.6. Every `scale-interval` timesteps until `scale-end`, the velocities are adjusted so that the kinetic energy corresponds exactly to the desired temperature (the value of control parameter `temperature`).

The control parameter `scale-options` selects which of the several possible variants of scaling are to be performed. This is an integer parameter interpreted as a set of bit flags with the following meanings.

bit 0 perform scaling for each molecular species individually.

bit 1 scale the rotational and translational components of the kinetic energy separately.

bit 2 use the rolling averages of kinetic energy to calculate the scale factor rather than the instantaneous values.

bit 3 discard all existing velocities and accelerations and re-initialize from the Maxwell-Boltzmann distribution.

These bits may be set in any combination so, for example `scale-options=6` sets bits 1 and 2 ($6 = 2^1+2^2$) and scales separately for rotation/translation using the rolling averages. If bit 3 is set the others are ignored.

[6] This is not quite true. *Moldy* does read the control file and any restart file but only to determine the name of the backup file. Thus even if the backup has a non-standard name it can still be found.

[7] This may be necessary if the restart file is located on a different device or disk partition from the current directory. To rename the temporary file successfully, it must reside in the same partition or device as the restart file

[8] From version 2.1 onwards

[9] Because the XDR calls are not part of ANSI standard C, however, the XDR code is conditionally compiled into *Moldy* only if the `USE_XDR` preprocessor symbol is defined during compilation.

inconsistency with the positions, which are of the centres of mass, *not* the zeroes of co-ordinates in the system specification file. This may be fixed in future releases.) Quaternions need only be included for polyatomic species, that is molecules 1 and n above, and omitted for the monatomic species i.

After the molecular positions and orientations have been set up, their velocities (and angular velocities if appropriate) are initialized. Their values are sampled at random from the Maxwell-Boltzmann distribution for the temperature T, as given by the control parameter `temperature`. This is done for both starting methods.

3.3 Restarting from a Previous Run

At the end of a simulation run, it is often desirable to store the configuration of the system in a file. This *restart file* may be used at a later date to continue the simulation from that point rather than from scratch. To instruct *Moldy* to write a restart file, simply set the control parameter `save-file` to a suitable filename; to start from a restart file, set `restart-file` to be the name of that file.

Each restart file is a binary file which contains enough information to reconstruct the exact state of the system and of the program. It includes a copy of all the control parameters in force, the current timestep number, a complete system specification, all the simulation dynamic variables and the intermediate data used in the calculation of averages and radial distribution functions. Thus a run continued from a restart file will proceed just as if there had been no interruption and will generate identical results (provided the control parameters are not changed).

When continuing a simulation, it is only necessary to explicitly specify control parameters which are to be changed. Their previous values are read from the restart file and are used as defaults when reading the control file. Consequently control files for restarting tend to be rather short. **Caution**: always include a new (possibly null) value for `save-file`. Otherwise when the new run terminates, the new restart file may overwrite the old one[4].

Neither is it necessary to repeat the system specification since that too is stored in the restart file. However there are occasions when it is desirable to do just that, for example if the value of one of the potential parameters is to be modified. In that case, set the switch `new-sys-spec` to 1 (true) and provide a system specification as per a new simulation. This is checked for consistency with the existing one and if correct replaces it. The following checks are applied, which only verify that it is sensible to assign the old dynamic variables to the new system. *1.* The number of species must be the same. *2.* Each species must have the same number of rotational degrees of freedom as its predecessor. It is not possible to replace a polyatomic by a monatomic or linear molecule, for example. *3.* The number of molecules of each species must not change. This means that the order in the specification file must be identical too. It is however possible to change the number of sites on a molecule, subject to *2.*

3.3.1 Periodic Backup

Closely related to restarting is the backup mechanism. This is provided to guard against the complete loss of a simulation due to computer failure. Periodically during a run, *Moldy* writes its state to a *backup file* – which is in fact just a restart file. In the event of a crash, the simulation can be restarted from the point the last backup was written rather than from the beginning. The related control parameters are `backup-file` which specifies the file name and `backup-interval` which gives the frequency of backups. It should not normally be necessary to change the name, but the optimum interval will depend on the size of the simulated system and the speed of the computer. By default it is 500. At the successful end of a run the backup file is deleted so that only if there is an abnormal termination does one remain[5].

The restart from a backup is entirely automatic. If a backup file exists when a run is started, it is read in and the run continues from it. In contrast to a normal restart all of the control parameters

[4] Whether the old file is lost depends on the operating system. Under systems such as VMS and COS which have version numbers a new version is created and the old one remains. Under Unix, the old file is renamed by the addition of a "%" character and thus is saved. On other systems it will be lost.

[5] A backup file is also written if the run is terminated for exceeding its cpu limit.

and has two parameters, $\epsilon(\equiv p^1_{ij})$ and $\sigma(\equiv p^2_{ij})$, which occur on each line in that order. Note that the definition of epsilon differs by a factor of 4 from another definition in common usage.

Buckingham This includes potentials of the Born-Meyer type and has formula

$$\phi(r_{ij}) = -A_{ij}/r^6_{ij} + B_{ij}\exp(-C_{ij}r_{ij}).$$

The three parameters appear on each line in the order A, B, C.

MCY This type supports potentials of the same form as the water model of Matsuoka, Clementi and Yoshimine[24],

$$\phi(r_{ij}) = A_{ij}\exp(-B_{ij}r_{ij}) - C_{ij}\exp(-D_{ij}r_{ij}),$$

and the four parameters appear on the line in the order A, B, C, D.

Other types of potential types may be easily added: see appendix 4.3.1.

It is possible to specify the units in which these quantities are given by means of the control file parameters `mass-unit`, `length-unit`, `time-unit` and `charge-unit` (which are themselves specified in SI units). All quantities read from the system specification file (dimensions as well as potentials) are taken to be in those units. Their default values are amu, Å, 0.1ps and q_e, which means that the unit of energy is kJ mol^{-1}. So to read in Å, amu and kcal mol^{-1}, specify `time-unit=4.8888213e-14`.

Once the system specification has been read in, all quantities are converted to 'internal' units: a.m.u., Å, ps, and $\sqrt{}$(a.m.u. Å^{3}ps^{-2}/$(4\pi\epsilon_0)$). The prototype molecule for each species is then shifted so that its zero of coordinates lies on its centre of mass, and rotated into the principal frame (polyatomics only).

3.2.2 The Initial Configuration

Moldy provides two methods of setting up an initial configuration. By default the *skew start* method of section 2.9.1 is used to place the molecular centres of mass in a regular arrangement which ensures molecular separation. If there is more than one species present, molecules of each are chosen randomly for each site. Molecular orientations are chosen randomly from a uniform distribution. This method has been found to work well for reasonably small or fairly isotropic molecules and it is anticipated that it will be the usual method of starting a simulation of the liquid state. On the other hand, if the constituent molecules are sufficiently large and irregular, or if it is intended to simulate the solid state then the *lattice start* method will be more appropriate.

This method is activated by setting the control parameter `lattice-start` to 1, and creates the initial configuration by periodic replication of some crystalline unit cell. In that case *Moldy* expects to find, following the `end` which terminates the system specification, an initial configuration specification of the following form:

$a\ b\ c\ \alpha\ \beta\ \gamma\ n_x\ n_y\ n_z$
$species\text{-}name_1\quad X_1\quad Y_1\quad Z_1\quad q_{10}\quad q_{11}\quad q_{12}\quad q_{13}$
$\vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots$
$species\text{-}name_i\quad X_i\quad Y_i\quad Z_i$
$\vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots\quad \vdots$
$species\text{-}name_n\quad X_n\quad Y_n\quad Z_n\quad q_{n0}\quad q_{n1}\quad q_{n2}\quad q_{n3}$
`end`.

Here a, b, c and α, β, γ are the crystal unit cell parameters, and n_x, n_y, n_z are the number of unit cells in each direction which comprise the MD cell. The next n lines describe the n molecules of the basis which will be replicated to form the full configuration. Molecules may appear in any order, but of course the total number of each, multiplied by the number of unit cells $n_x n_y n_z$ must agree with that given in the system specification file.

Each molecule is identified by its name, as given in the system specification file. X, Y and Z are *fractional* co-ordinates, between 0 and 1 giving the location of the molecular centres of mass in the crystal unit cell. The orientation is given by the four quaternions q_0, q_1, q_2, q_3 which specify a rotation *relative to the orientation of the prototype molecule in the system specification file.* (Notice the slight

Table 3.2: Control Parameters (continued)

name	type[a]	default	function
begin-average	integer	1001	When to start accumulating thermodynamic averages.
average-interval	integer	5000	How frequently to calculate and print averages.
reset-averages	boolean	false	Discard accumulated averages in restart file.
begin-rdf	integer	1000000	When to start accumulating radial distribution function information.
rdf-interval	integer	20	How frequently binning calculation is performed.
rdf-out	integer	5000	How frequently to calculate and print RDFs.
rdf-limit	real	10	Calculate RDFs up to what distance?
nbins	integer	100	Number of binning intervals between 0 and rdf-limit.
xdr	boolean	false	Write restart, backup and dump files in portable binary format using Sun XDR.
dump-file	character	null	Template of file names used for data dumps.
begin-dump	integer	1	Timestep to begin dumping at.
dump-interval	integer	20	How frequently to perform dumps.
dump-level	integer	0	Amount of information to include in dump.
ndumps	integer	250	Number of dump records in each dump file.
backup-interval	integer	500	Frequency to write backup file.
backup-file	character	MDBACKUP	Name of backup file.
temp-file	character	MDTEMPX	Name of temporary file used for writing restart configurations.
subcell	real	0	Size of sub cell to divide MD cell into for link-cell force calculation.
seed	integer	1234567	Seed for random number generator.
page-width	integer	132	Number of columns on output paper.
page-length	integer	44	Number of lines on a page of output.
mass-unit	real	1.6605655e-27	Unit of mass for system specification file.
length-unit	real	1e-10	Unit of length for system specification file.
time-unit	real	1e-13	Unit of time for system specification file.
charge-unit	real	1.6021892e-19	Unit of charge for system specification file.

[a] See section 3.1

Table 3.1: Control Parameters

name	type[a]	default	function
`title`	character	Test Simulation	A title to be printed on all output.
`nsteps`	integer	0	Number of MD steps to execute.
`cpu-limit`	real	1e20	Terminate run if excessive CPU time used.
`step`	real	0.005	Size of timestep
`sys-spec-file`	character	null	Name of system specification file. Appended to control file if null.
`lattice-start`	boolean	false	Switch for crystalline initial configuration.
`save-file`	character	null	File to save restart configuration in.
`restart-file`	character	null	File to read restart configuration from.
`new-sys-spec`	boolean	false	Read restart configuration with changed system specification.
`text-mode-save`	boolean	false	Write a portable "restart" file consisting of control, system specification and lattice start files.
`density`	real	1.0	Initial density. Used by *skew start* only to determine initial MD cell dimensions.
`scale-interval`	integer	10	Frequency of velocity scaling to required temperature.
`scale-end`	integer	1000000	When to stop scaling.
`scale-options`	integer	0	Select variations on scaling.
`temperature`	real	0	Temperature to set up initial configuration at and to scale to.
`const-pressure`	boolean	false	Whether to use Parrinello and Rahman constant stress.
`w`	real	100.0	Value of P & R mass parameter.
`pressure`	real	0	Applied pressure.
`strain-mask`	integer	200	bitmask controlling $\boldsymbol{h}$ matrix constraint.
`alpha`	real	0.3	α parameter for Ewald sum.
`k-cutoff`	real	2	Reciprocal space cut off distance in Å^{-1}.
`cutoff`	real	10	Direct space cutoff distance in Å.
`strict-cutoff`	boolean	false	Flag to select rigorous or cheap but approximate cutoff algorithm.
`surface-dipole`	boolean	false	Include De Leeuw & Perram term in Ewald sum.
`roll-interval`	integer	10	Period over which to calculate rolling averages.
`print-interval`	integer	10	How frequently to print normal output.

[a] See section 3.1

The `begin` and `end` parameters behave in a special fashion when the simulation is continued from a restart file; they are interpreted *relative* to the current timestep. Notice especially that `nsteps`, the number of timesteps is treated in this way.

A complete list of the parameters, their meanings and default values appears in table 3.1.

3.2 Setting up the System

3.2.1 System Specification

The information which describes to *Moldy* the system to be simulated and the interaction potentials is contained in a file known as the *system specification file*. This may be presented to *Moldy* in either of two ways: If the control file parameter `sys-spec-file` is null or absent, it is assumed to be appended to the end of the control file. Otherwise the it is read from the file whose name is the value of `sys-spec-file`.

This file is divided into two sections. First is the description of the molecules, atoms or ions, which is followed by the potential functions. As for the control file, the input is case independent and free format, but line structured. Blank lines, spacing and comments are ignored.

The physical description consists of a series of entries, one for each molecular species, terminated by the keyword `end`. The entry for species i should have the form

$$
\begin{array}{llllllll}
\textit{species-name}_i\ N_i & & & & & & & \\
id_1 & x_1 & y_1 & z_1 & m_1 & q_1 & name_1 \\
id_2 & x_2 & y_2 & z_2 & m_2 & q_2 & name_2 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
id_{n_i} & x_{n_i} & y_{n_i} & z_{n_i} & m_{n_i} & q_{n_i} & name_{n_i}
\end{array}
$$

where $species\text{-}name_i$ is the name of the molecule and N_i is the number of molecules of that type in the system. Each molecule has n_i atoms, one for each line in that group and each kind of atom is identified by a number id_i (the site id) which will be used to specify the appropriate potential parameters. Its co-ordinates are (x_i, y_i, z_i), its mass is m_i, its charge is q_i and its name is $name_i$. See Appendix A for some sample system specification files.

If there is more that one atom of any type (in the system - not just the same molecule) it is sufficient to identify it by its *id* (and the site co-ordinates!). If m_i, q_i or $name_i$ *are* given they must agree exactly with the previous values or an error will be signalled.

Site ids, masses and charges are all checked for 'reasonableness' and impossible values cause an error. The set of site ids does not have to start at 1 or be contiguous, but since this may indicate a mistake, a warning is issued.

Following the physical specification is the specification of the potential functions. This takes the form

$$
\begin{array}{llllll}
\textit{potential-type} & & & & & \\
i & j & p^1_{ij} & p^2_{ij} & \cdots & p^r_{ij} \\
k & l & p^1_{kl} & p^2_{kl} & \cdots & p^r_{kl} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
m & n & p^1_{mn} & p^2_{mn} & \cdots & p^r_{mn} \\
\texttt{end} & & & & &
\end{array}
$$

where *potential-type* is one of the keywords `Lennard-Jones`, `Buckingham` or `MCY` to identify the kind of potentials to be used, i, j, k, l, m, n are site ids and p^α_{ij} is the αth potential parameter between sites i and j. There should be one line for each distinct pair of site ids. If any pair is omitted a warning is issued and the parameter values are set to zero.

The meaning of the parameters for the currently defined potentials is as follows:

Lennard-Jones The potential is

$$\phi(r_{ij}) = \epsilon((\sigma/r_{ij})^{12}) - (\sigma/r_{ij})^6),$$

for writing the output to. Parameters in the control file are specified by entries of the form `keyword = value` which appear one to a line, terminated by the special keyword `end`. Spaces and blank lines are ignored as are comments (*i.e.* the remainder of a line following a `#` symbol) and keywords may be entered in upper or lower case. For example

```
title= Moldy example      # This is a comment

# The above blank line is ignored
nsteps = 1000
step=0.0005
restart-file = RESTART.DAT
end                       # The control file ends here
```

sets the title of the simulation to "Moldy example", the number of steps to execute to 1000, the timestep to 0.0005ps and supplies the name of a restart file to read from.

It is not necessary to specify all of the parameters on each run. Unless it is explicitly assigned a value in the control file, each parameter has a default value. This is either the value listed in table 3.1 or, in the case where the simulation is continuing from a restart file, the value it had on the previous run (see section 3.3). Parameters are read in sequence from the control file, and may appear more than once, in which case only the final instance is used.

The two most important parameters are `step` which sets the size of the simulation timestep (in ps), and `nsteps` which specifies the number of steps to perform. Together these control the length of time to be simulated. It is also possible to specify that a run should be terminated after a certain amount of computer time has been used - given by parameter `cpu-limit`. This will be particularly useful in batch mode systems, where the run is killed after some specified CPU time has elapsed. Setting `cpu-limit` to the maximum time allowed will cause *Moldy* to terminate the run *before* the limit is reached and write out a backup file (see section 3.3.1).

There are several kinds of parameters:

character strings Apart from `title` these are just file names *e.g.* `sys-spec-file`[3]. No checks are performed on the validity of the name (because *Moldy* has to work on many different computer systems), so if you make a mistake you are likely to get an error message to the effect that *Moldy* couldn't find the file you asked for. To remove a default value, just specify a null string *e.g.* `save-file = .`

booleans These are just switches which turn a feature off or on. '0' means off or false and '1' means on or true. The parameters `text-mode-save`, `new-sys-spec`, `surface-dipole` and `lattice-start` are booleans.

real parameters Several parameters are real numbers *e.g.* `step` which specifies the timestep. They may be entered in the usual floating point or scientific notation *e.g.* `step = 0.0005` or `step = .5e-3`, and are taken to be in the units given in table 3.1.

integer parameters Parameters such as `dump-level` take a numeric value, which should be an integer.

timestep-related parameters Several parameters govern when some calculation begins and ends and how frequently it is performed in between. These are known as "begin", "end" and "interval" parameters, but are really a special case of integer parameters. For example `begin-average`, `dump-interval` and `scale-end`. The calculation begins *on* the timestep specified on the `begin` parameter, occurs every `interval` timesteps thereafter and ends *after* the timestep specified by the `end` parameter. Setting the `interval` parameter to zero is the usual method of turning that calculation off.

[3]Note for COS users: There is a special filename syntax for permanent dataset names ID's and usernames under COS (as described in the C library reference manual). The forms allowed are variants of OWN/ID/PDN where OWN is the dataset owner, ID is the ID string and PDN is the dataset name. The OWN or ID strings may be omitted, defaulting to the job owner and null respectively. Unnecessary slashes may also be omitted so that, for example OWN//PDN and ID/PDN have the obvious meaning. If only a dataset name is present, and it is a valid local name, a local dataset is tried first. Failing that it is taken to be a permanent dataset name.

Chapter 3

Running Moldy

The way *Moldy* is invoked depends to some extent on the operating system, but usually by issuing the command `moldy`[1]. There are two optional arguments - the name of the control file (see section 3.1) and the output file (see section 3.5). If either is omitted, control input is read from the "standard input" which may be a terminal or a job command file depending on the operating system and circumstances, and the output is written to "standard output" which may be a terminal or batch job logfile[2]. Here are examples for VAX/VMS, Unix (tm) and COS (Cray Operating System), which assume that in each case the command has been set up to invoke *Moldy*. Under VMS the commands

```
$ moldy control.dat output.lis
$ moldy control.dat
```

will start *Moldy* which will read its input from `control.dat`. The output will be directed to the file `output.lis` in the first case and written to the terminal or batch log in the second. Under UNIX any of

```
% moldy < control > output.lis
% moldy control output.lis
% moldy control
```

will cause moldy to read from the file called `control` and in the first two examples to write its output to `output.lis`. The command-line interface for MS-DOS is very similar. Under COS (Cray Operating System) any of the control statements

```
MOLDY.
MOLDY,'CONTROL OUTPUT'.
MOLDY,' < CONTROL > OUTPUT'.
```

will invoke *Moldy* , which in the first example will read the control file from the job file following the first `/EOF` marker and send the output to `$OUT`. In the last two examples *Moldy* will read from the local dataset `CONTROL` (which might, for example, be created using `FETCH` or `ACCESS`) and create another local dataset called `OUTPUT`.

3.1 The Control File

The information needed to initiate and control a run of *Moldy* is specified in a file known as the *control file*. This contains the parameters governing the run *e.g.* the number of timesteps to be executed or the frequency of output, and the names of files to be used *e.g.* for reading a restart configuration from or

[1] On VMS, `moldy` may be defined as a foreign command by `$ moldy :== $mydisk:[mydir]moldy` For Unix(tm) and MS-DOS the executable file `moldy` or `MOLDY.EXE` should be placed in the shell search path (*e.g.* in the current directory). For COS the executable binary should be in a local dataset called `MOLDY`, which is often a permanent dataset, locally accessed by `ACCESS,DN=MOLDY`.

[2] Some operating systems (Unix and COS) allow *file redirection* whereby the standard input is associated with some file. This may also be used to supply the control file, provided that no command line parameter is given.

2.10.3 Charged Frameworks

A minor complication arises when using a framework which is has a non-zero net electric charge. Although the system as a whole may be electrically neutral, the omission of framework-framework interactions from the calculation also means that the $\boldsymbol{k} = 0$ term does not vanish. To see this examine equation 2.18. In the non-framework case the indices i and j in the terms

$$\left|\sum_{i=1}^{N} q_i \cos(\boldsymbol{k}.\boldsymbol{r}_i)\right|^2 = \sum_{i=1}^{N}\sum_{j=1}^{N} q_i q_j \cos(\boldsymbol{k}.\boldsymbol{r}_i)\cos(\boldsymbol{k}.\boldsymbol{r}_j)$$

run over all site pairs. If $\boldsymbol{k} = 0$ the squared sum is

$$\left|\sum_{i=1}^{N} q_i\right|^2 = 0$$

If a framework is present the formulation becomes

$$\left|\sum_{i=1}^{N} q_i \cos(\boldsymbol{k}.\boldsymbol{r}_i)\right|^2 - \left|\sum_{i=M+1}^{N} q_i \cos(\boldsymbol{k}.\boldsymbol{r}_i)\right|^2$$

assuming sites $M+1 \ldots N$ are the framework sites. On setting $\boldsymbol{k} = 0$ this reduces to

$$\left|\sum_{i=1}^{N} q_i\right|^2 - \left|\sum_{i=M+1}^{N} q_i\right|^2$$

It is therefore necessary to modify the charged-system term of equation 2.18 to

$$U_z = \frac{1}{8\pi\epsilon_0 V\alpha^2}\left\{\left|\sum_{i=1}^{N} q_i\right|^2 - \left|\sum_{i=M+1}^{N} q_i\right|^2\right\} \tag{2.44}$$

- The framework is fixed in space and is not allowed to rotate. Any rotation would of course destroy the 3D structure. For most purposes it is convenient to regard the framework as being at rest, hence translational motion is forbidden also.

- No interactions between sites on a framework molecule and on itself or any of its periodic images are evaluated. That is, framework-framework interactions, both potentials and forces, are systematically excluded from the real-space and reciprocal-space parts of the ewald sum, including the point and intra-molecular self terms of equation 2.18. (The exact modifications to equations 2.18,2.19 *etc* are left as an exercise for the reader.)

- In the real-space force calculation, sites are treated as being independent atoms rather than belonging to a molecule. In particular the cutoff is applied on the basis of the (framework) site to (fluid) molecule distance. By virtue of the "all-image" convention, all the requisite molecule-framework interactions are correctly included. When assigning sites to sub-cells, each site is therefore placed in the sub-cell which contains its co-ordinate. (By contrast sites belonging to an ordinary molecule are placed in the cell which contains the molecular centre of mass.)

With these modifications, *Moldy* is able to successfully model a fluid in contact with a 3D rigid framework. 2-dimensional systems such as a fluid at a surface or between two surfaces may be represented as a 3D system by adding an artificial periodicity in the third dimension. To reduce the errors so introduced, the framework can be made "large with space inside" to fill a MD cell with a large c-axis.

In the case of a 3D framework it is clearly not sensible to allow the MD cell to change shape or size, since this would destroy the internal structure of the framework. However if the framework represents a 2D layer or surface, then the layer spacing may be allowed to vary using the constant-stress equations (section 2.7) and applying constraints to allow only the c-axis to fluctuate. In that case, bear in mind that the dynamics are governed by the Parinello-Rahman equations of motion for the cell, rather than the Newtonian equations for the layer. In particular the mass is given by the parameter W rather than the mass of the framework molecule. (These may, of course be set equal if required.) Note also that no layer-layer interactions are calculated, and any force is the result of the externally applied pressure[4].

Finally there are two subtle complications which arise from the framework concept.

2.10.2 Stress and Pressure Undefined

There is no unique definition of the internal stress or pressure of a framework system. The pressure of a system in a space with periodic boundary conditions is defined in terms of the molecular virial

$$\mathcal{W} = \frac{1}{3}\sum_{i=1}^{N}\sum_{j\neq i}^{N} \boldsymbol{R}_{ij}.\boldsymbol{F}_{ij} \tag{2.42}$$

But the framework has no centre-of-mass, and so the quantity $\boldsymbol{R}_{ij}$ can not be defined. The site-virial formulation of equation 2.15 is of no assistance as the definition of the "internal" co-ordinate $\boldsymbol{p}_{i\alpha}$ involves the centre of mass co-ordinate $\boldsymbol{R}_i$. Neither can one simply choose a convenient reference $\boldsymbol{R}_i$. Since the force exerted by the fluid acting on the framework is in general non-zero, the term

$$\sum_i\sum_\alpha \boldsymbol{p}_{i\alpha}\boldsymbol{f}_{i\alpha} = \sum_i\sum_\alpha \boldsymbol{r}_{i\alpha}\boldsymbol{f}_{i\alpha} - \sum_i \boldsymbol{R}_i\boldsymbol{F}_i \tag{2.43}$$

clearly depends on $\boldsymbol{R}_i$. The situation may also be viewed physically. The definition of pressure of a system is the derivative of the free energy with respect to volume. But with an infinite rigid framework the volume derivative can not be defined.

The useful quantity in this case is the partial pressure of the fluid. Though not currently implemented, this may be rectified in a future version of *Moldy*.

Finally we note that in the case of a 2D layer structure, which is *not* rigid in the third dimension the perpendicular component if the stress tensor *does* have a physical meaning and is correctly calculated.

[4] This is a restriction of the current implementation and may be lifted in future versions.

which is ensured by assigning a random velocity

$$\omega_{ik}^{p} = \sqrt{\frac{k_B T}{I_{ik}}} R_{ik} \tag{2.39}$$

Since the dynamical variables used by *Moldy* for the angular co-ordinates are in fact the quaternions and their derivatives, we must set the quaternion derivatives and accelerations to the corresponding values. Using equations 2.9 and 2.10 we have

$$\dot{\mathbf{q}} = \mathbf{q}(0, \boldsymbol{\omega}^p/2) \tag{2.40}$$

and

$$\ddot{\mathbf{q}} = -\frac{1}{4}(\omega^p)^2 \mathbf{q} \tag{2.41}$$

Finally, we note that if a molecule has less than three degrees of freedom, that is $I_{ik} = 0$ for some i the corresponding angular velocities *etc* are simply set to zero.

2.10 Frameworks

In addition to the bulk properties of solids and liquids, much attention has been devoted in recent years to using MD simulation to model atoms or molecules interacting with surfaces or other structures such as zeolites. The distinctive feature of such a situation is the 2- or 3-dimensional surface or structure is larger than the interacting molecules by many orders of magnitude. In fact the idealization of this system makes the structure infinite in extent. An atomistic model of this kind of a system necessitates choosing the periodic boundary conditions of the MD cell to be commensurate with the periodicity of the surface or structure. This manual will refer to an infinitely repeating crystalline surface or structure as a *framework*.

There are two possible formulations of a system of this kind for use in a MD simulation. Firstly the framework may be modeled as a set of independent atoms interacting via pair potentials. This merely requires specifying the correct initial structure and MD cell plus a suitable set of pair potentials. The latter model both the internal forces which determine the crystal structure of the framework and its interaction with the atoms or molecules of the fluid. Conceptually there is no distinction between this situation and a regular solid or liquid system, and the mechanics of initiating a simulation are handled in exactly the usual manner.

However there are situations in which this "atomic" approach is impractical. Because the system being modeled is essentially "two-phase" the atoms of the framework find themselves under the influences of two distinct kinds of force. There are the strong forces, usually covalent or ionic, which bind the atoms to form the framework itself and the weaker, non-bonded forces of the interaction with the fluid. Ideally these could all be modeled by a single consistent set of interatomic potentials which are sufficiently transferable to yield an accurate crystal structure for the framework as well as the molecular structure of the fluid and its interaction with the surface. Regrettably such potentials are hard to find.

Furthermore the characteristic vibrational frequencies of the solid framework will probably my much higher than those of the fluid. Consequently the timestep must be chosen sufficiently small to accurately model the crystalline vibrations. This will usually be far smaller than the value required for the fluid, necessitating very lengthy MD runs to model both regimes properly. This is of course exactly the argument used to justify rigid-molecule models.

Moldy implements a variation on the rigid-molecule model to simulate rigid a framework structure periodically extended throughout space. There are a few subtleties which must be correctly handled to achieve a consistent implementation, which are described hereafter.

2.10.1 Implementation

The framework is in many respects exactly like an ordinary molecule. It should be defined to exactly fill the MD cell so that the periodic repeat of the cell generates the periodicity of its crystal structure. The distinctive features of a framework molecule are:

"inter-line" spacings a, d_y and d_z respectively

$$\begin{aligned} a &= \frac{L}{N}\sqrt{h^2+k^2+l^2} \\ d_y &= L\frac{\sqrt{k^2+l^2}}{\sqrt{h^2+k^2+l^2}} \\ d_z &\approx L\frac{l}{k} \qquad \text{(assuming } h/k \text{ is an integer)} \end{aligned} \tag{2.33}$$

The "equal spacing" requirements are satisfied approximately by

$$\begin{aligned} h &\approx N^{2/3} \\ k &\approx N^{1/3} \\ l &= 1 \end{aligned} \tag{2.34}$$

which when substituted into equation 2.33 yields

$$a \approx d_y \approx d_z \approx LN^{-1/3} \tag{2.35}$$

Using this method an arbitrary number of molecules may be packed into a cubic cell with a guaranteed minimum centre-of-mass separation of approximately $LN^{-1/3}$. In contrast to other methods, such as random placement with excluded volume, it will always yield a configuration no matter how high the density. It is also very simple to implement.

It still remains to determine the initial orientations of the molecules in the case of polyatomics. In the current implementation these are assigned randomly, which works well for small or near-spherical molecules. Further refinements which may help avoid overlaps for elongated molecules are possible, such as a periodic sequence of orientations along the line, but no investigation of this possibility have yet been carried out.

In practice the method has proved to work well for water and aqueous systems and always yields a configuration which may be evolved towards equilibrium by the MD equations of motion. Any feedback on its performance in more difficult cases will be welcome.

2.9.2 Initial Velocities

It remains to choose the initial velocities of the molecules to complete the specification of the initial configuration. The recipe is the same irrespective of whether the molecules are started on a lattice or from a skew start. The initial centre-of-mass velocities are chosen from the Maxwell-Boltzmann distribution at the temperature specified for the simulation[1, pp 170]. That is, the velocities are chosen from a set of random numbers with a Gaussian distribution and normalized so that the probability density $p(v)$ of the xyz component of the velocity v_{ik} of molecule k is

$$p(v_{ik}) = \left(\frac{m_k}{2\pi k_B T}\right)^{1/2} \exp(-\frac{m_k v_{ik}^2}{2k_B T}) \tag{2.36}$$

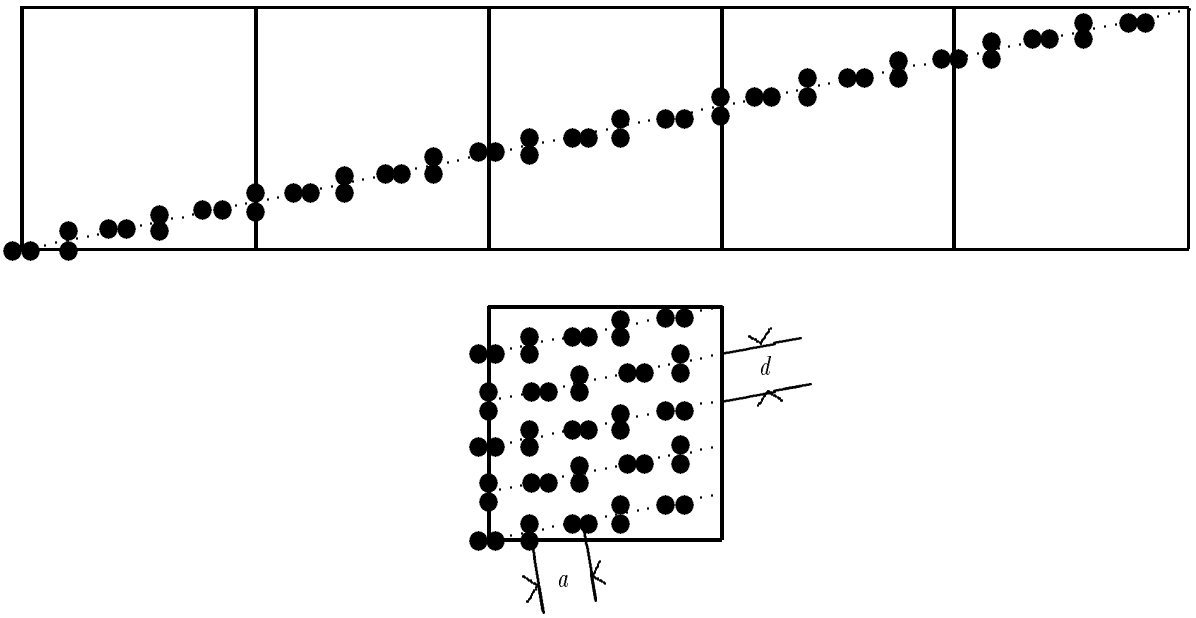
This is easily accomplished given a random number generator which samples from a Gaussian distribution with unit variance. Given a random number R_{ik}, each component of velocity is set to

$$v_{ik} = \sqrt{\frac{k_B T}{m_k}} R_{ik} \tag{2.37}$$

Each component of the angular velocity (expressed in the principal frame) has a probability distribution

$$p(\omega_{ik}^p) = \left(\frac{I_{ik}}{2\pi k_B T}\right)^{1/2} \exp(-\frac{I_{ik}(\omega_{ik}^p)^2}{2k_B T}) \tag{2.38}$$

Figure 2.1: The Skew Start method. N molecules are placed at evenly-spaced intervals, a on a line joining a corner of the MD cell to its image k cells away (5 in this illustration). When mapped back into the original cell this guarantees a minimum separation of $min(d, a)$.



some value. The timestep is chosen so that this criterion is satisfied for near-equilibrium configurations. But if the configuration is far from equilibrium certain forces may be extremely large (due to atoms approaching each other too closely). And worse, the breakdown of the integrator leads to breakdown of the conservation laws and the system evolves to a state even further from equilibrium.

One way around this difficulty is to start the system from a configuration known to be near-equilibrium, such as a known crystal structure. For a solid-state simulation this is the method of choice, and *Moldy* allows any initial structure to be specified and replicated to fill the MD cell. In the case of a liquid, a crystalline initial state is less desirable, and indeed, none may be known. Furthermore such a starting configuration restricts the allowed number of molecules to a multiple of the number in the unit cell and worse, may force the use of a non-cubic MD cell.

Moldy incorporates a novel method of generating an initial configuration which, in the main, overcomes these problems.

2.9.1 The Skew Start Method

The essence of the *Skew Start* method is to generate a configuration which, though not periodic in 3 dimensions, nevertheless is sufficiently regular to guarantee a minimum separation between molecular centres of mass. Figure 2.1 demonstrates the principle in 2 dimensions.

The N molecules are placed at some suitable interval a on a line drawn between one corner of the MD cell (of side L) and one of its periodic images. Clearly the index (h, k) of the image cell corner should be chosen so that the molecule spacing, a is close to the spacing of the line's images, d. For simplicity, choose $k = 1$ which leads to the condition:

$$\begin{aligned} a = \frac{L\sqrt{h^2+1}}{N} &\approx \frac{L}{\sqrt{h^2+1}} = d \\ \Rightarrow h &\approx \sqrt{N-1} \end{aligned} \tag{2.32}$$

Therefore the optimum value of h is the nearest integer to $\sqrt{N-1}$.

The formalism may be extended to three dimension, and yields the results for the molecular and

Moldy incorporates a rather different constraint which is not only simpler to implement (as it does not require modification of the equations of motion) but which also has a more obvious physical interpretation. The lower three sub-diagonal elements of the $\boldsymbol{h}$ matrix are constrained to zero. In other words the MD cell vector $\boldsymbol{a}$ is constrained to lie along the x-axis and $\boldsymbol{b}$ is constrained to lie in the xy-plane. Physically this may be thought of as implementing an MD box lying on a horizontal surface under the influence of a weak gravitational field. The implementation is trivial; at each timestep the acceleration of those components, $\ddot{\boldsymbol{h}}_{ij}$, is set to zero which is equivalent to adding a fictitious opposing force.

This constraint technique is not restricted merely to eliminating redundant degrees of freedom, but can also be used for other purposes. For example it may be used to allow uniaxial expansion only. The most important use of $\boldsymbol{h}$ matrix constraints however is probably for simulations of liquids. Since a liquid can not support shear stress there is no restoring force to keep the simulation cell nearly cubic, and it will therefore drift to a parallelepiped shape. To counter this tendency $\boldsymbol{h}$ may be constrained so that only the diagonal elements are non-zero and allowed to change. This does not give a completely isotropic MD cell expansion, but the time average should tend towards a cubic cell.

MD cell constraints are selected using the control parameter `strain-mask` (see section 3.8). A value of 238 will freeze the off-diagonal components of $\boldsymbol{h}$.

2.8 Radial Distribution Functions

The *radial distribution function* or RDF is one of the most important structural quantities characterizing a system, particularly for liquids. For a one-component system, the RDF is defined as[23, p445]

$$\begin{aligned} g(r) &= \frac{1}{\rho^2}\left\langle \sum_i \sum_{j\neq i} \delta(\boldsymbol{r}+\boldsymbol{r}_i-\boldsymbol{r}_j)\right\rangle \\ &\approx V\left\langle \delta(\boldsymbol{r}+\boldsymbol{r}_1-\boldsymbol{r}_2)\right\rangle \end{aligned} \tag{2.27}$$

where we use the angle brackets to denote a spherical average as well as the usual configurational average. Allen and Tildesley[1, pp184,185] describe how to evaluate $g(r)$ from a histogram of pair distances accumulated during a simulation run. With the notation that N is the total number of particles, b is the number of the histogram bins, δr is the bin width (so that $r = b\delta r$), $n_{his}(b)$ is the accumulated number per bin, τ is the number of steps when binning was carried out

$$g(r+\delta r/2) = \frac{3n_{his}(b)}{4\pi\rho N\tau[(r+\delta r)^3 - r^3]} \tag{2.28}$$

In the case of a molecular system, the partial RDF for atoms α and β is defined as[23, p 445]

$$g_{\alpha\beta}(r) = \frac{1}{\rho^2 V}\left\langle N(N-1)\delta(\boldsymbol{r}+\boldsymbol{r}_{1\alpha}-\boldsymbol{r}_{2\beta})\right\rangle \tag{2.29}$$

which may be rewritten more usefully for an arbitrary multi-component mixture by eliminating the molecular density ρ and number N as

$$g_{\alpha\beta}(r) = V\left\langle \delta(\boldsymbol{r}+\boldsymbol{r}_{1\alpha}-\boldsymbol{r}_{2\beta})\right\rangle \tag{2.30}$$

In the simulation this is evaluated by an expression very similar to equation 2.28

$$g_{\alpha\beta}(r+\delta r/2) = \frac{3Nn_{his}(b)}{4\pi\rho N_\alpha N_\beta\tau[(r+\delta r)^3 - r^3]} \tag{2.31}$$

2.9 The Initial Configuration

One of the more trying aspects of initiating a molecular dynamics simulation is getting it started in the first place. It is not hard to see the reason why. An MD integration algorithm will only generate a good approximation to the correct equations of motion if the forces and velocities of the particles are less than

to use the *average* kinetic energy $\langle \mathcal{K} \rangle$ as in equation 2.21, but this quantity is not known until after the run is completed. Because of this equation 2.22 can only set the temperature to an accuracy of $1/g$. This is often inadequate for purposes of comparison with experiment.

In order to allow the temperature to be set with greater accuracy, *Moldy* allows the use of a partial average in the denominator,

$$s = \sqrt{\frac{g k_B T_d}{2 \langle \mathcal{K} \rangle'}} \tag{2.23}$$

where $\langle \mathcal{K} \rangle'$ is the "rolling" average of $\mathcal{K}$ over some number of preceding timesteps. That number is determined by the control parameter `roll-interval`.

This option should be used cautiously. The change in $\langle \mathcal{K} \rangle$ upon scaling only has a significant effect on the average after many timesteps. If the subsequent scaling is performed before this change is reflected in the value of $\langle \mathcal{K} \rangle'$ it will use an out-of-date value of the average kinetic energy. It is therefore recommended that the number of timesteps between scalings be greater than or equal to the number used to compute the rolling average. Otherwise it is possible to produce wild overshoots and oscillations in the temperature.

Finally, there is a method for tackling really difficult cases when even individual scaling is unable to keep the temperature under control. This might be a far-from-equilibrium configuration where the potentials are so strong that velocities rapidly become very large, or when a single molecule acquires a very large velocity. In that case the velocities can all be re-initialized randomly from the Maxwell-Boltzmann distribution periodically. This provides a kind of pseudo monte-carlo equilibration for difficult cases.

2.7 Constant Stress

It is frequently desirable to conduct simulations under conditions of constant pressure or stress, rather than constant volume. For example, this allows the simulation of a solid-state phase transition with a change of symmetry or unit call size. *Moldy* incorporates the constant pressure method of Parrinello and Rahman[22].

In a constant-stress simulation the MD cell changes in size and shape in response to the imbalance between the internal and externally applied pressure. For an exposition of the method the reader should refer to Parrinello and Rahman's paper[22] and to Nosé and Klein's extension to rigid molecular systems[19]. The equation of motion for the reduced centre-of-mass so-ordinates $\boldsymbol{S}_i = \boldsymbol{h}^{-1}\boldsymbol{R}_i$ is

$$M_i \ddot{\boldsymbol{S}}_i = \boldsymbol{h}^{-1}\boldsymbol{F}_i - M_i \boldsymbol{G}^{-1}\dot{\boldsymbol{G}}\dot{\boldsymbol{S}}_i \tag{2.24}$$

replacing the straightforward Newton equation 2.3. $\boldsymbol{h}$ denotes the 3×3 MD cell matrix whose columns are the MD cell vectors $\boldsymbol{a}, \boldsymbol{b}$ and $\boldsymbol{c}$, $\boldsymbol{F}_i$ is the centre-of-mass force and $\boldsymbol{G} = \boldsymbol{h}'\boldsymbol{h}$. Equation 2.4, the Euler equation governs the angular motion exactly as in the constant-volume case.

The dynamics of the unit cell matrix $\boldsymbol{h}$ are governed by the equation

$$W\ddot{\boldsymbol{h}} = (\boldsymbol{\pi} - p)\,\boldsymbol{\sigma} \tag{2.25}$$

where W is a fictitious mass parameter, $\boldsymbol{\sigma} = V\boldsymbol{h}'^{-1}$ and p is the external pressure. The instantaneous internal stress $\boldsymbol{\pi}$ is given by

$$\boldsymbol{\pi} = \frac{1}{V}\sum_{i=1}^{N} m_i(\boldsymbol{h}_i\dot{\boldsymbol{s}}_i)^2 + \boldsymbol{\pi}^{s.r.} + \boldsymbol{\pi}^{e} \tag{2.26}$$

with the short-ranged and electrostatic components given by equations 2.15 and 2.20 respectively.

Nosé and Klein[19] describe and address the problem of the whole MD cell rotating during the course of the simulation. Angular momentum is not conserved in a periodic system, and because the $\boldsymbol{h}$ matrix has 9 degrees of freedom, three more than needed to specify the position and orientation of the MD cell. Their solution is to constrain the $\boldsymbol{h}$ matrix to be symmetric, and involves a modification of the Parrinello-Rahman equations.

twice the greatest molecular radius. This ensures that all appropriate interactions are included. Furthermore, all interactions between sites further apart than the cutoff are excluded (by the expedient of setting their separation to a large value in the potential calculation). This means that large and asymmetric molecules are handled correctly.

For a given cutoff radius the cell-based mode is rather quicker than the strict mode since the neighbouring cell list is much smaller and fewer interactions are computed. However to ensure that significant interactions are not omitted, it the cutoff ought to be set to a greater value than strictly required. This tends to offset the potential speed gain. On the other hand, if strict isotropy is required in a liquid simulation for example then the strict cutoff option ought to be used.

2.6 Temperature Initialization and Scaling

From the equipartition theorem, each degree of freedom in the system, f has the same kinetic energy, $\langle \mathcal{K} \rangle_f = \frac{1}{2} k_B T$. The effective temperature of the system is therefore given by the ensemble average of its kinetic energy.

$$T = \langle \mathcal{T} \rangle = \frac{2}{g k_B} \left\langle \sum_{f=1}^{g} \mathcal{K}_f \right\rangle = \frac{1}{3 N k_B} \left\langle \sum_{i=1}^{N} m_i \boldsymbol{v}_i^2 + \boldsymbol{\omega}_i . \boldsymbol{I} . \boldsymbol{\omega}_i \right\rangle \tag{2.21}$$

Here $\mathcal{K}_f$ is the instantaneous kinetic energy of degree of freedom f, g is the number of degrees of freedom, N is the number of molecules, $\mathcal{T}$ is an instantaneous "temperature".

It is almost always desirable that a simulation be conducted so that the temperature is the supplied parameter rather than the kinetic energy. This requires some mechanism to fix the *average* kinetic energy at thermal equilibrium. The *initial* kinetic energy may be set approximately by choosing random velocities which sample the Maxwell-Boltzmann distribution at the desired temperature, and this is indeed what *Moldy* does on starting a new run (see section 2.9.2). But because the initial configuration is usually far from equilibrium it will have too much potential energy. As the run progresses this will be converted into kinetic energy, raising the temperature above the desired value.

It is therefore necessary to have some mechanism for removing excess kinetic energy as the run progresses. *Moldy* uses the common technique of *velocity scaling*. At periodic intervals linear and angular velocities are multiplied by a factor of

$$s = \sqrt{\frac{g k_B T_d}{2 \mathcal{K}}} \tag{2.22}$$

where T_d is the desired temperature. By repeatedly setting the "instantaneous" temperature to the correct value while the system approaches its equilibrium state, the kinetic energy is made to approach its desired value. *Scaling* may be performed every timestep, or every few depending on the simulation conditions.

An MD run with scaling does not generate a valid statistical ensemble, and it must therefore be switched off before any calculation of thermodynamic averages is performed.

Moldy incorporates two refinements to the basic scaling algorithm (which are selected by the control parameter `scale-options`). Linear and angular velocities can be scaled independently, either for the whole system or for each species individually. In this way, one does not rely on the interactions of these degrees of freedom for convergence to equilibrium. In many systems these degrees of freedom are loosely coupled and the exchange of energy between them is slow. In these cases individual scaling can speed up the approach to equilibrium considerably.

The other refinement addresses the problem of setting the temperature accurately. At equilibrium the system's kinetic energy fluctuates with mean-square amplitude[3] $\langle \delta \mathcal{K}^2 \rangle = \frac{1}{2} g (k_B T)^2$, which corresponds to a rms fluctuation in the instantaneous "temperature" $\sqrt{(\langle \delta \mathcal{T}^2 \rangle)} = \sqrt{(2/g \langle \mathcal{T} \rangle)}$. The difficulty with applying equation 2.22 is the instantaneous kinetic energy $\mathcal{K}$ in the denominator. Strictly, scaling ought

[3] This formula actually applies to the Canonical rather than the microcanonical ensemble, but it serves for the purpose of this argument.

that only site pairs closer some *cutoff* distance r_c need be considered. Several methods are available to enumerate site pairs and choose those within the cutoff.

Most simple MD codes simply loop over all pairs of particles in the MD box and compute the separation r for each. If $r < r_c$ the interaction is computed. This method suffers from several disadvantages. Since for any given site, the interaction with any other site is considered only once, only the *nearest* periodic image of that site is included (the *minimum-image* convention). However this restricts the cutoff radius to less than half the MD cell dimension $r_c < 2L$. More serious is the way the computational time scales with the number of sites. If there are N sites, there are $O(N^2)$ separations to compute and the overall time therefore scales as $O(N^2)$.

The Verlet *Neighbour List* scheme[1, pp 146-149] makes use of the spatial locality of the interaction potential by maintaining a list for each site of all the "neighbouring" sites (*i.e.* all those within the cutoff distance). This can give considerable gains for moderate numbers of sites, but it ultimately requires $O(N^2)$ time (to build the lists) as well as $O(N)$ storage for the lists.

Moldy uses an implementation of the Link-Cell method of Quentrec *et al.*[20] described in Allen and Tildesley's book[1, pp 149-152] which is a true $O(N)$ algorithm. The fundamental idea is that the MD cell is partitioned into a number of smaller cells, known as *subcells*. Every timestep a linked list of all the particles contained in each subcell is constructed. The selection of all pairs of *particles* within the cutoff is achieved by looping over all pairs of *subcells* within the cutoff and particles within the subcells. Because of their regular arrangement, the list of "neighbouring" subcells is fixed and may be precomputed. Its construction takes only $O(N)$ operations and only $O(N)$ pair interactions need be calculated.

When the system consists of polyatomic molecules it is important that all sites belonging to a particular molecule are assigned to the same cell. Otherwise it is impossible to calculate the stress and pressure (equation 2.20) correctly as MD cell vectors are added to some intra-molecular distances. Therefore all sites of any molecule are assigned to a cell if the molecular centre-of-mass lies inside it.

One drawback of the link-cell method has been the difficulty of implementing it efficiently for vector processors. The linked list of "neighbour" particles is not stored in the regular-stride array which is required for vectorization. Heyes and Smith[21] pointed out that *gather* operations might be used to assemble a temporary array of neighbour particle co-ordinates from which the interaction potential and forces could be evaluated in vector mode. A *scatter* operation is then used to add the resulting forces to the total force array. This is the technique used in *Moldy*. Almost all modern vector machines have scatter/gather hardware which means these operations are fairly cheap.

2.5.1 No minimum-image convention

One notable feature of the implementation of the link-cell method in *Moldy* is that it does *not* follow the *minimum image* convention used in most MD codes. Instead the list of neighbouring cells, and hence interactions considered, includes *all* periodic images of a particle which are within the cutoff. This means that it is quite safe to use a cutoff of more than half of the MD cell side in any direction. By default the arrays are sized to allow a cutoff of up to unity times the MD cell side. This may be increased by increasing the constant `NSH` from 1 in `force.c`.

2.5.2 Cell or Strict cutoff

There are two options for the way in which site-site interactions are selected for inclusion in the total energy and forces. These are *cell-based* cutoff and *strict* cutoff and are selected by the `strict-cutoff` control parameter.

In cell-based mode (`strict-cutoff=0`) the neighbour cell list is built to include only those cells whose centre is within the cutoff radius of the centre of the reference cell. All interactions between sites belonging to molecules in the neighbouring cell list are computed. This is a "quick and dirty" method as some interactions between sites closer than the cutoff will inevitably be excluded whereas some outside the cutoff range will be included.

In strict mode (`strict-cutoff=1`) all interactions between pairs of *sites* within the cutoff are included. The neighbouring cell list contains all pairs of cells with any parts closer than the cutoff plus

that in that case the dipole moment per unit volume (or per MD cell) does *not* decrease with the size of the cluster.

The final term in equation 2.18 is just the dipole energy, and ought to be used in any calculation of the dielectric constant of a dipolar molecular system. It is switched on by *Moldy*'s control parameter `surface-dipole`. Note that as it represents the dipole at the surface of the cluster the system is no longer truly periodic.

Conversely it *must not* be used if the simulated system contains mobile ions. Consider an ion crossing a periodic boundary and jumping from one side of the MD cell to another. In that case the dipole moment of the MD cell changes discontinuously. Because of the surface dipole term the calculation would model a discontinuous macroscopic change in the dipole moment of the whole system caused by an infinite number of ions jumping an infinite distance. This is manifested in practice by a large and discontinuous change in the energy of the system and on the force on each charge within it.

This situation is completely non-physical but is easily avoided. However the problem may also arise more subtly even when there are no mobile ions if a framework is being simulated (section 2.10). The framework is treated as a set of discrete, but fixed atoms rather than a molecular unit. If the shape of the unit cell is allowed to vary then ions constituting the framework may indeed cross MD cell boundaries causing the aforementioned problems.

2.4.3 Stress

The internal stress (and pressure) of an atomic system is given by the volume-derivative of the internal energy. The situation is slightly more complicated for rigid molecules since molecules do not scale with volume and only the inter-molecular distances vary. The resulting expression for the coulombic part of the "instantaneous" stress π^e_{ik} is[19, Appendix A]

$$\begin{aligned} V\pi^e_{lm} &= \underbrace{\frac{1}{4\pi\epsilon_0}\sum_{\boldsymbol{n}}\sum_{i=1}^{N}\sum_{j=i+1}^{N} q_i q_j \left\{\frac{\operatorname{erfc}(\alpha|\boldsymbol{r}_{ij}+\boldsymbol{n}|)}{|\boldsymbol{r}_{ij}+\boldsymbol{n}|}+\frac{2\alpha}{\sqrt{\pi}}e^{-\alpha^2|\boldsymbol{r}_{ij}+\boldsymbol{n}|^2}\right\}\frac{(\boldsymbol{r}_{ij}+\boldsymbol{n})_l(\boldsymbol{r}_{ij}+\boldsymbol{n})_m}{|\boldsymbol{r}_{ij}+\boldsymbol{n}|^2}}_{\text{Real-space term}} \\ &+ \underbrace{\frac{1}{2\epsilon_0 V}\sum_{\boldsymbol{k}\neq 0}\frac{1}{k^2}e^{-\frac{k^2}{4\alpha^2}}\left(\delta_{lm}-2\left[\frac{1}{k^2}+\frac{1}{4\alpha^2}\right]\boldsymbol{k}_l\boldsymbol{k}_m\right)\left\{\left|\sum_{i=1}^{N}q_i\cos(\boldsymbol{k}.\boldsymbol{r}_i)\right|^2+\left|\sum_{i=1}^{N}q_i\sin(\boldsymbol{k}.\boldsymbol{r}_i)\right|^2\right\}}_{\text{Reciprocal-space term}} \\ &- \underbrace{\sum_{i=1}^{N}(\boldsymbol{F}_i)_l(\boldsymbol{p}_i)_m}_{\text{Molecular Virial Correction}} + \underbrace{\frac{\delta_{lm}}{8\pi\epsilon_0 V\alpha^2}\left|\sum_{i=1}^{N}q_i\right|^2}_{\text{charged system term}} \end{aligned} \tag{2.20}$$

The true internal stress is the ensemble average average, $\pi_{lm} = \left\langle \pi^e_{lm} + \pi^{s.r.}_{lm} + \pi^K_{lm} \right\rangle$, where $\pi^{s.r.}_{lm}$ and π^K_{lm} are the short-range force and kinetic contributions respectively. π^e_{lm} enters into the Parrinello-Rahman equations of motion (see section 2.7).

The term marked *Molecular Virial Correction* in equation 2.20 is the difference between the site-site virial $\sum_i f_i.r_i$ and the molecular virial $\sum_i F_i.R_i$ and is subtracted after all of the site forces and the molecular forces have been calculated including the short-range potential components which are not included in the equations above. Though is is not apparent in reference[19, Appendix A] this term has exactly the same form for all parts of the stress — the short-range potential and the real- and reciprocal space coulombic parts.

2.5 Periodic Boundaries — the Link Cell Method

The real space part of the Ewald (equation 2.18) and short-range potential energy is a sum of contributions over pairs of sites. In both cases the interaction decays rapidly with separation, which means

$$= \underbrace{\frac{1}{4\pi\epsilon_0}\sum_{\boldsymbol{n}}\sum_{j\neq i}^{N} q_j \left\{ \frac{\mathrm{erfc}(\alpha|\boldsymbol{r}_{ij}+\boldsymbol{n}|)}{|\boldsymbol{r}_{ij}+\boldsymbol{n}|} + \frac{2\alpha}{\sqrt{\pi}} e^{-\alpha^2|\boldsymbol{r}_{ij}+\boldsymbol{n}|^2} \right\} \frac{\boldsymbol{r}_{ij}+\boldsymbol{n}}{|\boldsymbol{r}_{ij}+\boldsymbol{n}|^2}}_{\text{Real-space term}}$$

$$+ \underbrace{\frac{1}{\epsilon_0 V}\sum_{\boldsymbol{k}\neq 0} q_i \frac{\boldsymbol{k}}{k^2} e^{-\frac{k^2}{4\alpha^2}} \left\{ \sin(\boldsymbol{k}.\boldsymbol{r}_i)\sum_{j=1}^{N} q_j \cos(\boldsymbol{k}.\boldsymbol{r}_j) + \cos(\boldsymbol{k}.\boldsymbol{r}_i)\sum_{j=1}^{N} q_j \sin(\boldsymbol{k}.\boldsymbol{r}_j) \right\}}_{\text{Reciprocal-space term}} \qquad (2.19)$$

$$+ \underbrace{\left[\frac{q_i}{6\epsilon_0 V} \left(\sum_{j=1}^{N} q_j \boldsymbol{r}_j \right) \right]}_{\text{surface dipole term}}$$

Both the real- and reciprocal-space series (the sums over $\boldsymbol{n}$ and $\boldsymbol{k}$) converge fairly rapidly so that only a few terms need be evaluated. We define the *cut-off* distances r_c and k_c so that all terms with $|\boldsymbol{r}_{ij}+\boldsymbol{n}| < r_c$ and $\boldsymbol{k} < k_c$ are included.

The molecular forces and torques $\boldsymbol{F}$ and $\boldsymbol{N}$ are evaluated from the site forces $\boldsymbol{f}_i$ using equations 2.1 and 2.2.

Notice that the equation 2.18 for the energy contains a correction for the intra-molecular self-energy, whose derivative is absent from the equation for the forces (equation 2.19). This term corrects for interactions between charges on the *same* molecule which are implicitly included in the reciprocal space sum, but are not required in the rigid-molecule model. Though the site forces $\boldsymbol{f}_i$ do therefore include unwanted terms these sum to zero in the evaluation of the molecular centre-of-mass forces and torques (equations 2.1 and 2.2) (by the conservation laws for linear and angular momentum).

2.4.1 Uniform Sheet Correction

The 5th term in equation 2.18 necessary only if the system has a nonzero net electric charge, and is useful in special cases such as framework systems.

In a periodic system the electrostatic energy is finite only if the total electric charge of the MD cell is zero. The reciprocal space sum in equation 2.18 for $\boldsymbol{k} = 0$ takes the form

$$\frac{1}{k^2} e^{-\frac{k^2}{4\alpha^2}} \left| \sum_{i=1}^{N} q_i \right|^2$$

which is zero in the case of electroneutrality but infinite otherwise. Its omission from the sum in equation 2.18 is physically equivalent to adding a uniform jelly of charge which exactly neutralizes the unbalanced point charges. But though the form of the reciprocal space sum is unaffected by the uniform charge jelly the real-space sum is not. The real-space part of the interaction of the jelly with each point charge as well as the self-energy of the jelly itself must be included giving the fifth term in equation 2.18.

2.4.2 Surface Dipole Term

The optional final term in equations 2.18 and 2.19 if used performs the calculations under different periodic boundary conditions. It was suggested by De Leeuw, Perram and Smith[18] in order to accurately model dipolar systems and is necessary in any calculation of a dielectric constant.

The distinction arises from considerations of how the imaginary set of infinite replicas is constructed from a single copy of the MD box[1, pp 156-159]. Consider a near-spherical "cluster" of MD cells. The "infinite" result for any property is the limit of its "cluster" value as the size of the cluster tends to infinity. However this value is non-unique and depends on the dielectric constant, ϵ_s of the physical medium surrounding the cluster. If this medium is conductive ($\epsilon_s = \infty$) the dipole moment of the cluster is neutralized by image charges, whereas in a vacuum ($\epsilon_s = 1$) it remains. It is trivial to show

In addition to the short-range potential electric charges may be specified for each atomic site which interact according to Coulomb's Law. See the following section for details.

2.4 The Ewald Sum

The long range Coulomb interactions are handled using the Ewald Sum technique in three dimensions[17, 1, p. 156]. The electrostatic potential of a system of charges is expressed as a sum of short-range and long-range contributions. Each term is written as a series,the first in real space and the second, obtained by Fourier transformation using the periodicity of the MD cell, in reciprocal space. The expression for the Coulomb energy U is

$$\begin{aligned} U &= \underbrace{\frac{1}{4\pi\epsilon_0}\sum_{\boldsymbol{n}}\sum_{i=1}^{N}\sum_{j=i+1}^{N} q_i q_j \frac{\mathrm{erfc}(\alpha|\boldsymbol{r}_{ij}+\boldsymbol{n}|)}{|\boldsymbol{r}_{ij}+\boldsymbol{n}|}}_{\text{Real-space term}} \\ &+ \underbrace{\frac{1}{2\epsilon_0 V}\sum_{\boldsymbol{k}\neq 0}\frac{1}{k^2}e^{-\frac{k^2}{4\alpha^2}}\left\{\left|\sum_{i=1}^{N} q_i\cos(\boldsymbol{k}.\boldsymbol{r}_i)\right|^2+\left|\sum_{i=1}^{N} q_i\sin(\boldsymbol{k}.\boldsymbol{r}_i)\right|^2\right\}}_{\text{Reciprocal-space term}} \\ &- \underbrace{\frac{\alpha}{4\pi^{\frac{3}{2}}\epsilon_0}\sum_{i=1}^{N} q_i^2}_{\text{Point self-energy}} \underbrace{-\frac{1}{4\pi\epsilon_0}\sum_{n=1}^{M}\sum_{\kappa=1}^{N_m}\sum_{\lambda=\kappa+1}^{N_m} q_{n\kappa}q_{n\lambda}\frac{\mathrm{erf}(\alpha|\boldsymbol{r}_{\kappa\lambda}|)}{|\boldsymbol{r}_{\kappa\lambda}|}}_{\text{Intra-molecular self energy}} \\ &+ \underbrace{\frac{1}{8\pi\epsilon_0 V\alpha^2}\left|\sum_{i=1}^{N} q_i\right|^2}_{\text{charged system term}} + \underbrace{\left[\frac{1}{6\epsilon_0 V}\left|\sum_{i=1}^{N} q_i\boldsymbol{r}_i\right|^2\right]}_{\text{surface dipole term}} \end{aligned} \tag{2.18}$$

where we are using the notation

$\boldsymbol{n}$	Lattice vector of periodic array of MD cell images.
$\boldsymbol{k}$	Reciprocal lattice vector of periodic array of MD cell images..
k	Modulus of $\boldsymbol{k}$.
i, j	absolute indices of all charged sites.
n	index of molecules.
κ, λ	indices of sites within a single molecule.
N	Total number of charged sites.
M	Total number of molecules.
N_m	Number of sites on molecule m.
$\boldsymbol{p}_i$	Co-ord of site i relative to molecular centre-of-mass, $\boldsymbol{r}_i - \boldsymbol{R}_i$.
q_i	Charge on absolute site i.
$q_{m\kappa}$	Charge on site κ of molecule m.
$\boldsymbol{r}_i$	Cartesian co-ordinate of site i.
$\boldsymbol{r}_{ij}$	$\boldsymbol{r}_j - \boldsymbol{r}_i$.
α	Real/reciprocal space partition parameter.
π_{lm}	Instantaneous stress tensor.
δ_{lm}	Dirac delta function.
l, m	xyz tensor indices.
V	Volume of MD cell.

and the force on charge i is given by

$$\boldsymbol{f}_i = -\nabla_{\boldsymbol{r}_i} U$$

respective molecule. The total potential energy of the system is

$$U = \sum_i \sum_{j>i} \sum_\alpha \sum_\beta \phi_{i\alpha j\beta}(\boldsymbol{r}_{i\alpha j\beta}). \tag{2.14}$$

where $\boldsymbol{f}_{i\alpha j\beta} = -\boldsymbol{\nabla}\phi_{i\alpha j\beta}(\boldsymbol{r}_{i\alpha j\beta})$ is the force acting on site β of molecule j from site α of molecule i.

The total force and torque acting on any particular molecule are calculated using equations 2.1 and 2.2. Since $\boldsymbol{f}_{i\alpha j\beta}$ and therefore $\boldsymbol{F}_{ij}$ are short-ranged forces (*i.e.* they decay faster than r^{-3}) one can define a *cutoff* radius, r_c. Interactions between sites whose separation r_{ij} is greater than r_c are assumed to be negligible and need not be evaluated. In the case of a polyatomic molecular system it is usual to apply the cutoff according to the *intermolecular* separation R_{ij}.

2.3.1 Pressure and Stress

The internal stress of a system of interacting molecules is given by Allen and Tildesley[1, pp 46-49] in terms of the molecular virial, but we may rewrite it more conveniently in terms of the atomic site virial as

$$\begin{aligned} V\boldsymbol{\pi}^{sr} &= \left\langle \sum_i M_i \boldsymbol{V}_i \boldsymbol{V}_i + \sum_i \sum_{j>i} \boldsymbol{R}_{ij} \boldsymbol{F}_{ij} \right\rangle \\ &= \left\langle \sum_i M_i \boldsymbol{V}_i \boldsymbol{V}_i + \sum_i \sum_{j>i} \sum_\alpha \sum_\beta \boldsymbol{r}_{i\alpha j\beta} \boldsymbol{f}_{i\alpha j\beta} - \sum_i \sum_\alpha \boldsymbol{p}_{i\alpha} \boldsymbol{f}_{i\alpha} \right\rangle \end{aligned} \tag{2.15}$$

The quantity $\boldsymbol{p}_{i\alpha} \equiv \boldsymbol{r}_{i\alpha} - \boldsymbol{R}_i$ is the co-ordinate of each site relative to the molecular centre-of-mass. The pressure is easily evaluated as one third of the trace of the stress tensor.

2.3.2 Long Range Corrections

The truncation of the interactions at the cut-off radius does introduce some errors into the calculated potential energy and stress. By neglecting density fluctuations on a scale longer that the cutoff radius we may approximate the errors and calculate correction terms[1, pp 64-65].

$$U_c = \frac{2\pi}{V} \sum_\alpha \sum_\beta N_\alpha N_\beta \int_{r_c}^\infty r^2 \phi_{\alpha\beta}(r) \mathrm{d}r \tag{2.16}$$

$$\boldsymbol{\pi}_c = \frac{1}{V} U_c \boldsymbol{\delta} \tag{2.17}$$

where the sums run over all the distinct kinds of sites, N_α is the total number of sites of type α in the system and $\boldsymbol{\delta}$ is the unit matrix.

2.3.3 Potential Functions

Moldy supports most common forms of potential function. In particular,

Lennard-Jones	$\phi(r) = \epsilon((\frac{\sigma}{r})^{12}) - (\frac{\sigma}{r})^6)$
6-exp	$\phi(r) = -\frac{A}{r^6} + B\exp(-Cr)$ This is the most general 6-exp form and includes potential of the *Born-Meyer* and *Buckingham* forms.
MCY	$\phi(r) = A\exp(-Br) - C\exp(-Dr)$
generic	$\phi(r) = A\exp(-Br) + C/r^{12} - D/r^4 - E/r^6 - F/r^8$ This is a composite which contains terms of almost any type likely to be encountered, and is really for experimental use.

which is just the q_0 component of equation 2.9. Just as with the normalization condition, the integration algorithm will not preserve this condition exactly unless explicit measures are taken. After each timestep the constraint may be re-established by subtracting the discrepancy from the quaternion derivatives. If $\delta = q_0\dot{q_0} + q_1\dot{q_1} + q_2\dot{q_2} + q_3\dot{q_3}$ then the corrected quaternion derivatives are given by

$$\dot{\mathbf{q}}' = \dot{\mathbf{q}} - \delta\mathbf{q}. \tag{2.12}$$

Experiments conducted while developing *Moldy* show that enforcing this constraint significantly decreases the fluctuations in the total energy.

Linear molecules are a slightly special case as the moment of inertia about the molecular axis is zero. Though there are unique methods to represent this situation[1, page 90] *Moldy* uses a minor modification of the quaternion algorithm. All that is necessary is a little special-case code to avoid dividing by the zero component of inertia in the solution of equation 2.4 and to hold the components of angular velocity and acceleration about the molecular axis to zero. This has the considerable advantage of uniform treatment of all kinds of molecules which is convenient when dealing with heterogeneous mixtures.

2.2 Integration Algorithms

The dynamical equations 2.3 and 2.4 are integrated using this author's modification[13] of the Beeman algorithm[14]. For atomic systems it is equivalent to the commonly used Verlet algorithm[15] but is preferable for polyatomic molecules because it gives velocities accurate to order δt^3 compared to order δt^2 for the Verlet algorithm. This not only permits a more accurate determination of the kinetic energy, pressure and other dynamic quantities, but is essential if the (generalized) velocities enter the dynamical equations themselves.

Velocity-dependent forces occur in equations 2.4 and 2.10 and in the Parrinello-Rahman constant-pressure equations (section 2.7). These usually present a problem to non "predictor-corrector" algorithms which are based on the assumption that the forces depend only on the co-ordinates. Fincham has devised a scheme to allow integration of the rotational equations using Verlet-like algorithms[16], which is widely used despite the potential problems caused by the low accuracy of the velocities being propagated into the dynamics. The case of the Parrinello-Rahman equations has been handled by reformulating them to avoid the velocity-dependence[**?**].

However both cases are easily and accurately handled by the modification to Beeman's equations proposed by the author[13]. These may be summarized using the symbol x to represent any dynamic variable (centre-of-mass co-ordinate, quaternion or MD cell edge), $\dot{x}^{(p)}$ and $\dot{x}^{(c)}$ to represent "predicted" and "corrected" velocities respectively.

$$\begin{array}{lll}
i & x(t+\delta t) = x(t) + \delta t\,\dot{x}(t) + \frac{\delta t^2}{6}\left[4\ddot{x}(t) - \ddot{x}(t-\delta t)\right] & \\
ii & \dot{x}^{(p)}(t+\delta t) = \dot{x}(t) + \frac{\delta t}{2}\left[3\ddot{x}(t) - \ddot{x}(t-\delta t)\right] & \\
iii & \ddot{x}(t+\delta t) = F(\{x_i(t+\delta t), \dot{x}_i^{(p)}(t+\delta t)\}, i = 1\ldots n) & \left.\begin{array}{l}\\\\\\\end{array}\right\} \\
iv & \dot{x}^{(c)}(t+\delta t) = \dot{x}(t) + \frac{\delta t}{6}\left[2\ddot{x}(t+\delta t) + 5\ddot{x}(t) - \ddot{x}(t-\delta t)\right] & \\
v & \text{Replace } \dot{x}^{(p)} \text{ with } \dot{x}^{(c)} \text{ and goto iii. Iterate to convergence} &
\end{array} \tag{2.13}$$

The predictor-corrector cycle of steps *iii* to *v* are iterated until the predicted and corrected velocities have converged to a relative precision of better than 1 part in 10^{-7}, which in practice takes 2 or 3 cycles. This iteration is not as inefficient as it might at first appear as it does *not* include the expensive part of the calculation — the recalculation of the site forces. Only the angular accelerations and quaternion second derivatives must be evaluated at each iteration using equations 2.4 and 2.10, and this operation is relatively cheap.

2.3 Potentials and Short-Range Forces

The forces determining the dynamics of the system are derived from the potential function denoted by $\phi_{i\alpha j\beta}(\boldsymbol{r}_{i\alpha j\beta})$. The indices i and j run over all molecules in the system and α and β over sites on the

A quaternion is an ordered number quartet which obeys the algebra given by Pawley[5]. The multiplication rule in that reference may be restated as a matrix product treating each quaternion as a 4-vector. If $\mathbf{p} \equiv (p_0, p_1, p_2, p_3)$ and $\mathbf{q} \equiv (q_0, q_1, q_2, q_3)$ are quaternions then

$$\mathbf{pq} = \begin{pmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \tag{2.5}$$

The quaternion $\tilde{\mathbf{q}}$ conjugate to $\mathbf{q}$ is defined as $\tilde{\mathbf{q}} = (q_0, -q_1, -q_2, -q_3)$ so that

$$\mathbf{q}\tilde{\mathbf{q}} = (q_0^2 + q_1^2 + q_2^2 + q_3^2, 0, 0, 0). \tag{2.6}$$

The *norm* is defined as $|\mathbf{q}| \equiv \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$ and $\mathbf{q}$ is called a *unit* quaternion if $|\mathbf{q}| = 1$.

Any possible rotation can represented by a unit quaternion. Du Val shows[6] that if $\mathbf{q} = (\cos\frac{\alpha}{2}, \boldsymbol{l}\sin\frac{\alpha}{2})$ (where we have combined the last three components to form a 3-vector) and $\mathbf{p} = (0, \boldsymbol{r})$ then the operation

$$\mathbf{p}' \equiv (0, \boldsymbol{r}') = \mathbf{qp}\tilde{\mathbf{q}} \tag{2.7}$$

corresponds to a rotation of the vector $\boldsymbol{r}$ by an angle of α about the axis $\boldsymbol{l}$. The components may also be expressed in terms of the Euler angles as[2]

$$\begin{aligned} q_0 &= \cos\frac{\phi+\psi}{2}\cos\frac{\theta}{2} \\ q_1 &= \sin\frac{\phi-\psi}{2}\sin\frac{\theta}{2} \\ q_2 &= \cos\frac{\phi-\psi}{2}\sin\frac{\theta}{2} \\ q_3 &= \sin\frac{\phi+\psi}{2}\cos\frac{\theta}{2}. \end{aligned} \tag{2.8}$$

The relationship between the time derivative of a quaternion and the principal frame angular velocity was given by Evans[2, Equation 27] and rewritten using quaternion algebra by Refson[11] as

$$2\dot{\mathbf{q}} = \mathbf{q}(0, \boldsymbol{\omega}^p) \tag{2.9}$$

The second derivative is given by

$$\begin{aligned} 2\ddot{\mathbf{q}} &= \mathbf{q}(-1/2(\omega^p)^2, \dot{\boldsymbol{\omega}}^p) \\ &= \mathbf{q}(-2|\dot{\mathbf{q}}|^2, \dot{\boldsymbol{\omega}}^p) \end{aligned} \tag{2.10}$$

Equations 2.10 and 2.4 allow the simulation to be implemented using quaternions and their derivatives as the dynamic variables for rotational motion, and this is the method employed in *Moldy*. This second order formulation was first used by Powles *et al.*[12] and Sonnenschein showed[7] that it gives substantially better stability than if angular velocities and accelerations are used as dynamic variables.

Using equations 2.10 to describe the dynamics means that they are integrated as if all four components were independent. Therefore the normalization condition $\mathbf{q}\tilde{\mathbf{q}} = \mathbf{1}$ may not be exactly satisfied after performing an integration step. *Moldy* adopts the usual practice of scaling all components of the quaternion after each timestep to satisfy the normalization condition[3].

It is less widely realized that the second order equations (2.10) introduce a *second* "unconstrained" variable into the procedure. Differentiating equation 2.6 gives a constraint on the quaternion derivatives

$$q_0\dot{q_0} + q_1\dot{q_1} + q_2\dot{q_2} + q_3\dot{q_3} = 0 \tag{2.11}$$

[2]The definition of quaternions used here differs from that used in Evans' paper[2, equation 21] in the sign of q_2 or ξ. This error has been compounded by subsequent authors[7, 8, 9] who also managed to permute the components which means that the parameters do not form an ordered number quartet which obeys quaternion algebra. Like Allen and Tildesley[1, page 88] we follow the definition of of Goldstein[10, pages 143 and 155].

Chapter 2

Algorithms and Equations

This chapter describes the implementation of the molecular dynamics technique as used in *Moldy*. It is not intended as an introduction to MD simulations, and does assume some familiarity with the basic concepts of microscopic models and simulations thereof. The book by Allen and Tildesley[1] is a very good introductory text, covering both the theory and the practicalities and is highly recommended. It also contains comprehensive references to the scientific literature of microscopic computer simulation.

Moldy is designed to simulate a common class of models of atomic or molecular systems. The assumptions are: that the system is an assembly of *rigid molecules*, atoms or ions; that the forces of interaction are derived from *continuous potential functions* acting between (usually atomic) *sites* on each molecule; that the dynamics are governed by the *classical* Newton-Euler equations of motion. A major aim of *Moldy* has been to allow the most general of models within that class and to impose as few restrictions as possible. In particular arbitrary mixtures of different molecules are allowed and several popular forms of potential functions are catered for.

2.1 The Equations of Motion

If we denote the force exerted by atom α of molecule i on atom β of molecule j by $\boldsymbol{f}_{i\alpha j\beta}$[1]. then the total force acting on molecule i is

$$\boldsymbol{F}_i = \sum_j \sum_\beta \sum_\alpha \boldsymbol{f}_{i\alpha j\beta} \tag{2.1}$$

and the torque is given by

$$\boldsymbol{N}_i = \sum_\alpha (\boldsymbol{r}_{i\alpha} - \boldsymbol{R}_i) \times \boldsymbol{f}_{i\alpha} \tag{2.2}$$

where $\boldsymbol{R}_i = 1/M_i \sum_\alpha m_{i\alpha} \boldsymbol{r}_{i\alpha}$ is the centre of mass of molecule i.

The motion is governed by the Newton-Euler equations

$$M_i \ddot{\boldsymbol{R}}_i = \boldsymbol{F}_i \tag{2.3}$$

$$\boldsymbol{I}_i \cdot \dot{\boldsymbol{\omega}}_i - \boldsymbol{\omega}_i \times \boldsymbol{I}_i \cdot \boldsymbol{\omega}_i = \boldsymbol{N}_i \tag{2.4}$$

where $\boldsymbol{\omega}_i$ is the angular velocity of the molecule, $\boldsymbol{I}_i = \sum_\alpha m_{i\alpha} (p_{i\alpha}^2 \mathbf{1} - \boldsymbol{p}_{i\alpha} \boldsymbol{p}_{i\alpha})$ is the inertia tensor and $\boldsymbol{p}_{i\alpha} = \boldsymbol{r}_{i\alpha} - \boldsymbol{R}_i$ is the atomic site co-ordinate relative to the molecular centre of mass.

The orientations of the molecules are represented by *quaternions* as has now become common practice. They are preferred over Euler angles for two reasons. Firstly they lead to equations of motion which are free of singularities[2] which means that no special-case code is required. This leads to much improved numerical stability of the simulation[3]. Secondly, molecular symmetry operations and combinations of rotations are elegantly expressed in terms of a simple quaternion algebra[3, 4].

[1] A comment on notation is appropriate here. In this chapter, site quantities are denoted by *lowercase* letters, molecular quantities by *uppercase*, sites are indexed by *greek* letters and molecules by *roman*. A missing index denotes a sum over the corresponding sites or molecules so that, for example $r_{i\alpha j\beta}$ is a site-site vector and F_{ij} the molecule-molecule force

Chapter 1

Introduction

Moldy is a computer program for performing molecular dynamics simulations of condensed matter. It can handle any assembly of rigid polyatomic molecules, atoms or ions and any mixture thereof. It uses the 'link cell' method to calculate short-range forces and the Ewald sum technique to handle long-range electrostatic forces. Simulations may be performed either in the usual NVE ensemble or in an $N\sigma T$ ensemble using the constant-stress method of Parrinello and Rahman. As the MD cell need not be cubic, the program is equally suitable for simulations of solids and liquids.

Most existing MD programs are limited in their capabilities, for example to one kind of potential function, or molecular symmetry, or to some restricted number of molecules. *Moldy* is (as far as possible) free from such arbitrary constraints. The system is specified at the beginning of each run and its size is only limited by the amount of memory available to the program: if a system is too large to handle, the solution is to buy some more memory. The system may contain a mixture of an arbitrary number of molecular species, each with an arbitrary number of atoms and a arbitrary number of molecules of each. Molecules or ions may be monatomic or polyatomic, linear or three dimensional in any combination. The potential functions may be of the Lennard-Jones, Buckingham (including Born-Meyer) or MCY types, and other potential types may be easily added. Such flexibility is possible because *Moldy* is written in the 'C' language which permits dynamic memory allocation.

Moldy is written to be highly portable and has been tested on a wide range of computers and operating systems, including CRAY COS, VAX/VMS, MS-DOS and Unix[1] (both BSD and system V varieties). It should be straightforward to move it to any other machine with a good 'C' compiler.

To be of real use a simulation program must run efficiently on modern high-speed computers, which are increasingly of vector or parallel architectures. *Moldy* is written so as to be highly vectorizable and has been tested on a range of vector machines from manufacturers including Cray, Convex, Stardent and Alliant. On the cray XMP-48 its performance can exceed 100 MFlop/sec (on a suitably large system). A experimental version for parallel machines with shared memory is also available.

[1] Goodness knows who will own the Unix registered trademark by the time you read this.

3.11 Messages and Errors . . . 33

4 Compiling and Modifying Moldy **34**
4.1 Compilation . . . 34
4.2 Portability . . . 35
4.2.1 System Dependencies . . . 35
4.2.2 Optimization and Vectorization . . . 35
4.3 Modifying Moldy . . . 37
4.3.1 Adding a New Potential . . . 37
4.3.2 Parallel Version . . . 37

A Example System Specifications **39**
A.1 Argon . . . 39
A.2 TIPS2 Water . . . 39
A.3 Aqueous $MgCl_2$ Solution . . . 39

B Organization of Moldy **41**
B.1 Data Structures . . . 41
B.2 Control Flow . . . 41

C Utility Programs **42**
C.1 Moldyext . . . 42
C.2 Dumpanal . . . 42
C.3 Dumpconv . . . 42
C.4 Dumpext . . . 43
C.5 Mdshak . . . 44

Contents

1 Introduction **3**

2 Algorithms and Equations **4**
2.1 The Equations of Motion 4
2.2 Integration Algorithms 6
2.3 Potentials and Short-Range Forces 6
2.3.1 Pressure and Stress 7
2.3.2 Long Range Corrections 7
2.3.3 Potential Functions 7
2.4 The Ewald Sum 8
2.4.1 Uniform Sheet Correction 9
2.4.2 Surface Dipole Term 9
2.4.3 Stress 10
2.5 Periodic Boundaries — the Link Cell Method 10
2.5.1 No minimum-image convention 11
2.5.2 Cell or Strict cutoff 11
2.6 Temperature Initialization and Scaling 12
2.7 Constant Stress 13
2.8 Radial Distribution Functions 14
2.9 The Initial Configuration 14
2.9.1 The Skew Start Method 15
2.9.2 Initial Velocities 16
2.10 Frameworks 17
2.10.1 Implementation 17
2.10.2 Stress and Pressure Undefined 18
2.10.3 Charged Frameworks 19

3 Running Moldy **20**
3.1 The Control File 20
3.2 Setting up the System 22
3.2.1 System Specification 22
3.2.2 The Initial Configuration 25
3.3 Restarting from a Previous Run 26
3.3.1 Periodic Backup 26
3.3.2 Portable Restart Files 27
3.4 Setting the Temperature 27
3.5 Output 28
3.6 Radial Distribution Functions 29
3.7 Dumping 30
3.8 Constant Stress Simulation 31
3.9 Cutoffs and Adjustable Parameters 32
3.10 Framework Simulations 32

Moldy User's Manual

Revision: 2.6

Keith Refson
Department of Earth Sciences
Parks Road
Oxford OX1 3PR
keith@earth.ox.ac.uk

February 21, 1994