MD code “mdmorse”

- A simplified MD code `mdmorse` has been written for this course:

  - `mdmorse` simulates atom motion in a variety of metals (but only one metal at a time) with a simple Morse pair potential model.

  - The code has a Verlet neighbour list (but not a linked list) and the equations of motion are solved with the velocity Verlet method.

  - The code is given in Fortran90 and C.

- The code can be downloaded from the WWW home page of the course:

`http://beam.helsinki.fi/~knordlun/atomistiset/`

- But right now the physically interesting subroutines have been removed from the code, so it does not work.

- During the next few exercise sessions, you get the task of writing the missing subroutines. Solutions will be provided and explained during the exercise sessions. You may either use your own or the provided solutions afterwards (but be sure to check that you have some working version, otherwise the rest of the exercises will get screwed up).

- Like all codes, the `mdmorse` code may have bugs. **Anyone who first finds a serious bug in the code, gets 10 extra exercise points !**

Structure of the mdmorse code

Program files

main.f90	Main program
inout.f90	Miscellaneous input and output stuff
modules.f90	Global variables
physical.f90*	Calculating T and E, and random number generators
neighbourlist.f90*	Getting the neighbour list
solve.f90*	Solving the equations of motion
forces.f90*	Calculating the forces.
Makefile	Makefile

* These are the subroutines which will be filled up during the exercises

C version: as above but *.c instead of *.f90. And modules.f90 replaced by global.h. C version also uses the C++ comment character "//". On some C compilers it may be necessary to change this to the standard format /* text */

Compiling the code

```
make
```

- This has been tested to work at least in Alpha and Linux systems with a f90 compiler and the gcc compiler

Input files

mdmorse.in	Miscellaneous parameters
atoms.in	Atom coordinates (default: 5x5x5 FCC Cu)

Output files

(standard output)	T, Epot and other interesting output
atoms.out	Atom coordinates at regular intervals

Running the program:

./mdmorse

- Should be done in the same directory where the input files are

- Note that it may be necessary to increase the stack size on systems where the default is ridiculously low:

```
limit stacksize 20000 kb
```

- Note also that during the program running, the code writes out a large number of atom coordinates to a file atoms.out, which may grow very large. The code is worth running e.g. under /scratch/ so that you do not run out of disk space. Remember to remove old outfiles.

Input file mdmorse.in

```
Sample input file for mdmorse md program
 File format: -identifier, then value. Rest is arbitrary comments
 Lines which do not begin with "-" are all ignored

Identifier      | Value

-initialT        600.0            Initial temperature

-desiredT        300.0            Variables for temperature control
-btctau          0.0              If btctau=0 no effect

-bpctau          0.0           Variables for pressure control
-bpcbeta         7.0e-4           If bpctau=0 no effect
-desiredP        0.0

-mass            63.546           For Cu

-xsize           18.126900793     Box size in each dimension
-ysize           18.126900793
-zsize           18.126900793

-periodicx       1                1 = periodic, 0 = non
-periodicy       1
-periodicz       1

-morseDe         0.3429           Morse potential parameters for Cu
-morsealpha      1.3588
-morseRe         2.866

-rpotcut         5.0              Potential cutoff
-rskincut        6.0              Neighbour list cutoff, must be > rpotcut

-nupdate        5               Number of steps between neighbour list updates

-nmovieoutput    100              Interval between atom movie output

-deltat          2.0              Time step in simulation in fs

-tmax            10000.0          Total simulation time
```

Note that the parameters `desiredT, btctau, bpctau, bpcbeta` and `desiredP` can be ignored in the first stage of the exercices!

Input file atoms.in

- The file is a normal XYZ atom coordinate file:

```
500
FCC cell made by makeFCC with a= 3.615 n= 5 5 5
Cu     -8.13375     -8.13375     -8.13375 1
Cu     -6.32625     -6.32625     -8.13375 1
```

.... and so forth 496 atom coordinates....

```
Cu      6.32625      8.13375      8.13375 1
Cu      8.13375      6.32625      8.13375 1
```

- Note that the cell is centered on the origin !

Standard output

- When the code is working (not yet), the standard output will look something like the following:

```
--------------- mdmorse03 V1.2 --------------------

Read in parameter -initialT        value  600.000
Read in parameter -desiredT        value  600.000
Read in parameter -btctau          value  300.000
Read in parameter -bpctau          value  3000.00
Read in parameter -bpcbeta         value 0.700000E-03
Read in parameter -desiredP        value 0.000000
Read in parameter -mass            value  63.5460
Read in parameter -xsize           value  18.1269
Read in parameter -ysize           value  18.1269
Read in parameter -zsize           value  18.1269
Read in parameter -periodicx       value  1.00000
Read in parameter -periodicy       value  1.00000
Read in parameter -periodicz       value  1.00000
Read in parameter -morseDe         value 0.342900
Read in parameter -morsealpha      value  1.35880
Read in parameter -morseRe         value  2.86600
Read in parameter -rpotcut         value  5.00000
Read in parameter -rskincut        value  6.00000
Read in parameter -nupdate         value  5.00000
Read in parameter -nmovieoutput    value  100.000
Read in parameter -deltat          value  2.00000
Read in parameter -tmax            value  10000.0
Using periodics (1=on, 0=off) 1 1 1
Morse potential parameters: De alpha Re    0.342900    1.358800    2.866000
Movie output selected every     100 steps

Doing Berendsen temperature control with tau T   300.000   600.000


Doing Berendsen pressure control with tau beta  3000.000     0.001
```

```
Reading in     500 atoms described as FCC cell made by makeFCC with a= 3.62538
 Initial atom temperature is   1182.27222623179
Neighbour list update found    78.000     neighbours per atom
ec      2.000   1182.272     0.15117    -3.03933    -2.88816   171.18459
Outputting atom movie at t =      2.000
ec      4.000   1160.705     0.14730    -3.03603    -2.88872   171.70862
bpc  18.127868 18.127868 18.127868  5957.17235   171.70862     1.00003
ec      6.000   1122.386     0.14137    -3.03061    -2.88924   172.60850
ec      8.000   1068.872     0.13363    -3.02370    -2.89008   173.72576
ec     10.000   1002.443     0.12439    -3.01613    -2.89173   174.84455
```

- And so on. Here most things are self-explanatory. The “ec” lines contain the physically most interesting stuff in the format:

```
ec      time(fs) T(K)        Ekin/atom     Epot/atom   Etot/atom
```

Output file atoms.out

- The output file atoms.out is also an XYZ file, but with the exception that column 5 contains the atom kinetic energy, and column 6 the potential energy. So it will eventually look like:

```
500
mdmorse atom output at time      2.000 fs boxsize     18.1269    18.1269
18.1269
Cu    -8.148674   -8.155407   -8.146573   0.150744  -3.029602
Cu    -6.333828   -6.350433   -8.165532   0.178643  -3.039851
Cu    -8.151044   -6.335727   -6.339057   0.114739  -3.053847
```

Testing the incomplete code

Even though the code is not complete, it should compile and run in the intermediate stages.

The output should look something like:

```
Reading in     500 atoms described as FCC cell made by makeFCC with a= 3.62538
 Initial atom temperature is   0.000000000000000
Neighbour list update found   0.26928E+06 neighbours per atom
ec      2.000      0.000     0.00000     0.00000     0.00000     0.00000
Outputting atom movie at t =      2.000
ec      4.000      0.000     0.00000     0.00000     0.00000     0.00000
```

i.e. the number of neighbours is crazy, and the temperature is 0. When you start doing the exercises, this should change and interesting things will start to happen.

PS. Note on old versions

If you have access to old versions of the code, it is not a good idea to copy the old solutions directly, for two reasons: a) they are not compatible - I have made some changes which could result in some bugs which are very hard to track down if old solutions are copied directly, and b) I have *all* the old solutions, and may keep a lookout for suspiciously familiar new solutions.

After this we will go through the (incomplete) source code itself during the lecture.