DPDMACS

Michael Patra
patra@lorentz.leidenuniv.nl

There exist many programs for coarse-grained computer simulations, usually in the framework of DPD (dissipative particle dynamics). Unfortunately, available programs suffer from two limitations. First, they are not adapted to current computer architectures, and, second, they use their own proprietary input files, and learning how to use these programs thus can be very time-consuming.

DPDMACS is specially adapted to current x86-architectures, using handwritten code with assembler intrinsics for maximal speed. Running in 64 bit can increase the speed even further. All input and output files are compatible with the Gromacs suite for atomistic molecular dynamics simulations, thereby minimising the effort to learn and use DPDMACS.

I. INSTALLATION

DPDMACS is designed to be used on x64 architectures with the SSE and SSE2 extensions. This includes the Intel Pentium-4 and the AMD64 cpu's. Running in 64-bit mode is recommanded as this doubles the number of available general-purpose and floating-point registers. The program can be compiled and run also on other cpu architectures by means of the `xmm_func.h` header file but this will severely affect the speed.

Using the Intel C compiler is recommanded but any recent version of GCC will do, at the expensive of reducing the speed of DPDMACS by about $5\,\%$. Installation is straight-forward:

```
./configure
make
make install
```

Compiling textsfDPDMACS puts every compiler under stress due to its extensive innerloops. If you need to choose a different compiler because the default compiler fails to compile, you can specify the compiler in the `CC` environment variable:

```
./configure CC=/usr/local/bin/gcc
```

The `benchmark` subdirectory contains a few sample simulations that can be used to check the correct compilation.

II. USAGE INFORMATION

DPDMACS is designed to be as compatible as possible with the Gromacs suite of programs for atomistic molecular dynamics. We will shortly summarise the basic setup employed

by Gromacs, please see the Gromacs website www.gromacs.org for deeper information. Each simulation needs three different input files:

- initial coordinates (and initial velocities, if so desired). The file name has the extension .gro
- a "topology" file that describes the forcefield that is to be used for the simulation. File extension: .top
- an MD parameter that contains information such as the step size or the temperature. File extension: .mdp

The topology file frequently is rather long. For this reason, C preprocessor commands such as #include can be used to structure the file. It is a convention to put pieces of topologies into files with the extension .itp .

The result of the simulation is a trajectory file, extension .trr , that contains snapshots of the coordinates and velocities of all particles during the simulation.

Parsing the .top topology file can take quite a bit of time. For this reason, Gromacs also creates a binary topology file with the extension .tpr . This file is used by the analysis tools supplied with the Gromacs suite. For example, you want to compute a mass density profile from your simulation. The .trr trajectory file will contain the coordinates of the particles but not their masses – this information, the masses, will then be read by the mass density calculation program from the .tpr binary topology file.

DPDMACS follows the same approach: It reads in a .gro , a .top and a .mdp file and will create a .trr and a .tpr file. The format of the input files is identical to the original Gromacs format, there are just some minor extensions to allow to specify DPD parameters. In the following, we will describe the format of the input files, first for readers that are already familiar with Gromacs, and then for readers which are new to this subject.

A. Initial coordinate file (.gro)

1. *Readers familiar to Gromacs*

The only difference to the way that Gromacs handles this file type is that the atom numbers are actually checked (to see whether they are consecutive) whereas Gromacs silently ignores these numbers.

2. *Readers new to Gromacs*

A simple .gro file looks like this:

```
Test  system
    2
    1POL       B1     1    1.000     1.000     1.000
```

```
    1POL        B2      2    2.000     7.000     1.000
  10.00000    10.00000    10.00000
```

The first line just specifies some "name" for the system but is ignored otherwise. The second line gives the total number of particles. Then follows one line per particle.

The first field tells that this is the first molecule of type `POL` whereas the second field specifies the names of the two particles in that molecule, namely `B1` and `B2`. The third field gives the consecutive number (or index) of the particle. Finally, there are x, y and z coordinate of the particle in units of nanometres. Optinally, there could three additional fields specifying the velocities of the particles (in units of nanometres per picosecond).

At the end of the file, the size of the simulation box is given. If there would be two polymers present instead of just a single one, the file could look like this:

```
Test  system
    4
    1POL        B1      1    1.000     1.000     1.000
    1POL        B2      2    2.000     7.000     1.000
    2POL        B1      3    1.000     2.000     6.000
    2POL        B2      4    2.000     8.000     6.000
  10.00000    10.00000    10.00000
```

The position of the fields is relevant, i. e., whitespace does matter. You thus need to make sure to insert the correct number of space characters. Using one of the example files as template can be helpful. If you want to create `.gro` files from some program, the Fortran formatting command is

```
write(  unit=*,   fmt='(I5,2A5,I5,3F8.3,3F8.3)'         ) &
      moleculeNumber,     moleculeName,     particleName,     &
      particleNumber,     positionX,    positionY,    positionZ,    &
      velocityX,    velocityY,    velocityZ
```

and the C formatting string is

```
printf(   "%5d%-5s%-5s%5d%8.3f%8.3f%8.3f%8.3f%8          .3f%8 .3f\n ",
          moleculeNumber,     moleculeName,     particleName,
          particleNumber,     positionX,    positionY,    positionZ,
          velocityX,    velocityY,    velocityZ    );
```

If you have your coordinates available in some other more-or-less standard file format, you can try the `babel` utility, which you can download from `www.eyesopen.com/babel/`, or its successor OpenBabel, available from `openbabel.sourceforge.net`. They can convert from almost any file format to almost any other. In particular, you can use the `.xyz` file format as input which is rather straight forward.

If you are able to create `.pdb` files in some easy way, you can directly use the utilities from the Gromacs suite and simply type

```
editconf   -f  input.pdb   -o  output.gro
```

There is one minor issue when dealing with different file formats: some (like `.gro`) specify all positions in units of nanometres whereas others (like `.pdb`) use Ångström. Converting between two such file formats will thus change all numbers by a factor 10. This is precisely what is needed if you work with "real" systems where all lengths are specified in "real physical" units, like when each blob of a PEO polymer has a diameter of 0.4 nm. There thus is no reason for you to do anything special as everything is taken care of automatically. If, on the other hand, you want to use "generic" units where the diameter of the beads forming a polymer is simply set to 1, you need to watch out for this factor 10 that appears in the conversion. If you only want to convert between `.pdb` and `.gro` , this factor can be taken care of by `editconf` :

```
editconf   -f  input.pdb   -scale   10  10  10  -o  output.gro
editconf   -f  input.gro   -scale   0.1  0.1  0.1  -o  output.pdb
```

B. Topology file (`.top`)

1. *Readers familiar to Gromacs*

In the `[ defaults ]` section at the very beginning of the topology file, the combination rule is specified, e. g.,

```
[ defaults   ]
  1               3                 no                1.0       1.0
```

Combination rule "1" in Gromacs (geometric mean of c_6 and c_{12}) does not make sense for soft potentials. Combination rules "2" and "3" have the same meaning as in Gromacs, and an additional combination rule "4" has been introduced for completeness:

rule "2": $a = \sqrt{a_1 a_2}$ $\quad \sigma = \frac{1}{2}(\sigma_1 + \sigma_2)$
rule "3": $a = \sqrt{a_1 a_2}$ $\quad \sigma = \sqrt{\sigma_1 \sigma_2}$
rule "4": $a = \frac{1}{2}(a_1 + a_2)$ $\quad \sigma = \frac{1}{2}(\sigma_1 + \sigma_2)$

In DPD, no interactions between bounded pairs are excluded, so the other parameters on this line are ignored.

The DPD soft potential has the functional form

$$V(r) = \frac{a}{2}(\sigma - r)^2 \ . \tag{1}$$

It is specified by the new interaction type "3", and the two parameters are (in this sequence) σ (in units of nanometre) and a [in units of $\mathrm{kJ}/(\mathrm{mol}\cdot\mathrm{nm}^2)$]. To specify interactions between two particles of type `C` to be described by $\sigma = 1$ nm and $a = 5\ \mathrm{kJ}/(\mathrm{mol}\cdot\mathrm{nm}^2)$, you can use

```
[ nonbond_params     ]
    C C     3      1.0  5.0
```

The first line of a topology file should start with the line

```
[ defaults   ]
  1               3                    no                   1.0       1.0
```

The number "3" specifies the default combination rule while the other parameters on this line have no meaning and are just needed for compatibility with Gromacs. For each particle, the DPD interaction parameters a and σ need to be specified later on in the topology file, and the combination rule sets how the interaction between pairs of different particles should be computed. (Of course, you can overrule this rule for certain pairs.). Instead of the number "3" you can also specify "2" or "4":

rule "2": $a = \sqrt{a_1 a_2}$ $\quad \sigma = \frac{1}{2}(\sigma_1 + \sigma_2)$
rule "3": $a = \sqrt{a_1 a_2}$ $\quad \sigma = \sqrt{\sigma_1 \sigma_2}$
rule "4": $a = \frac{1}{2}(a_1 + a_2)$ $\quad \sigma = \frac{1}{2}(\sigma_1 + \sigma_2)$

After this, you need to specify the particle types that you want to use in your simulation:

```
[ atomtypes    ]
    C1      5.000        1.000        A   4.0  0.0
    C2      5.000        2.000        A   4.0  0.0
    C3      5.000        3.000        A   4.0  0.0
```

This defines three particle types, namely with the names `C1`, `C2` and `C3`. Each has mass $m = 4$ u (u is the atomic mass unit) and is uncharged (final column). The DPD interaction is of the form

$$V(r) = \frac{a}{2}(\sigma - r)^2 \ . \tag{2}$$

The second column specifies the parameter σ in units of nm, i. e., $\sigma = 1$ nm for all three particle times. The third column specifies a in units of $\mathrm{kJ}/(\mathrm{mol} \cdot \mathrm{nm}^2)$.

The interaction between a pair of two particles, let us say, one of type `C1` and one of type `C2`, is determined by the combination rule. With the default setting "3", this would give $\sigma = 1$ nm and $a = \sqrt{2}$ kJ/(mol · nm^2). You can overrule this with an optional `[ nonbond_params ]`. For example, this will make the interaction between `C1` and `C2` stronger and more long-ranged [$\sigma = 1.5$ nm and $a = 5.0$ kJ/(mol · nm^2)]:

```
[ nonbond_params     ]
    C1  C2     3       1.5  5.0
```

The number "3" is needed for compatability with Gromacs and must not be changed.

In a similar way, parameters are specified for bonds between atoms, and for angular potentials:

```
[ bondtypes    ]
   C1  C3          1      1.5  10.0

[ angletypes    ]
   C1  C2  C1      1      120  2.0
```

The number "1" specifies harmonic potentials. No other potentials are currently implemented. The last two numbers give the equilibrium bond length in nm and the force constant [in units of $\mathrm{kJ}/(\mathrm{mol}\cdot\mathrm{nm}^2)$], respectively the equilibrium angle in degrees and the force constant [in units of $\mathrm{kJ}/(\mathrm{mol}\cdot\mathrm{degrees}^2)$].

After that each molecule needs to be defined. Every particle in the simulation needs to belong to a molecule. Even if the molecule consists only of a single particle, you still need to define a molecule. The definition of a molecule starts with the name of the molecule.

```
[ moleculetype    ]
SomeMolecule    0
```

The number "0" specifies that no nonbonded interactions are excluded. For traditional MD simulations, frequently the nonbonded interactions with the two or three nearest neighbours are excluded. This is different for DPD simulations, and "0" thus needs to be specified here.

```
[ atoms  ]
1        C1      1        RES      A1       1     0.0
2        C2      1        RES      A2       2     0.0
3        C2      1        RES      A3       3     0.0
```

The first field simply counts the particles in the molecule, and the second field give the particle type, as defined further up in the `[ atomtypes ]` section.

The next two fields (`1` and `RES`) technically speaking give the residue number and residue name. This nomenclature was developed for proteins, where a single peptide molecule consists of many residues (=amino acids). This nomenclature can be applied also to other molecules, such as polymers, but this is basically only to make the files easier to understand for humans – it does not make any difference for the program.

The only important thing to remember in this context is that it are the residue name and residue number that appear in a `.gro` -file. The format string in Sec. II A 2 called it molecule name but actually it is the residue name. Hence, for the example, one has to use `RES` and not `SomeMolecule` :

```
    1RES       A1     1    2.000     7.000     1.000
```

The fifth field in the topology file is the name under which the particle is referenced in the `.gro` -file. Here, the names are thus `A1`, `A2` and `A3`.

The last number on each line gives the partial charge of the particles. In the example, all particles thus are uncharged. The number before it specifies the charge group and is ignored for DPD simulations.

After all atoms are specified, you need to specify which particles are connected by bonds.

```
[ bonds  ]
  1 2    1
  2 3    1
```

The first two numbers give the index numbers of the two particles, as specified in the [atoms] section. In this example, the bonds are thus A1–A2 and A2–A3. The number "1" at the end of the line means that a harmonic potential is to be taken. The parameter for the bond potential are taken from the [bondtypes] section further above. It is, however, also possible to overrule those settings here:

```
[ bonds  ]
  1 2   1
  2 3   1    2.5  10.0
```

The same then has to be done for angular potentials.

```
[ angles  ]
  1 2 3    1
```

This defines an angular potential that depends on the angle A1–A2–A3 (as specified by the first three numbers) and that is harmonic (as specified by the "1" at the end). Note that all pairs of particles that are mutually connected need to have an entry under [bonds], otherwise the result would be very strange. However, in many cases there will be no angular potentials, e. g., when describing freely jointed polymers.

At this point in the file, all properties of the particles and the molecules are fully specified. Now, all that remains is to decide how the system that you want to simulate, is composed. First, you have to specify some title will be used as comment in some of the created output files, but otherwise it has no function.

```
[ system  ]
This  is  just  some  name  that  you  like
```

The final thing that is left is to specify how many molecules of which molecule type should be included in the simulation.

```
[ molecules   ]
SomeMolecule          10
DifferentMolecule      5
```

The order in which the molecules are given here, has to be the same as the order of the coordinates in the .gro -file.

C. MD parameter file (.mdp)

The topology .top -file defines the potentials between the particles in the simulation, and the coordinate .gro -file defines the initial conditions. The MD parameter file specifies what should actually be done, like "run a simulation for so many steps at such a temperature…". This file consists of a number of lines of the form

```
parameter    = value
```

The order of lines in this file is irrelevant. DPDMACS supports the relevant parameter entries from GROMACS, with a few extensions to treat the special needs of DPD simulations. We will describe these extensions first, and will then describe the most important parameters copied from GROMACS for readers new to this subject.

1. Extensions to the parameter file

`tcoupl: [andersen — lowe ]`

Selecting `andersen` will active an Andersen thermostat that will at random times (as determined by `tau _t`) assign a random new velocity (as determined by `ref _t`) to an individual particle.

Selecting `lowe` will active a Lowe-Andersen thermostat that will at random times (as determined by `tau _t`) make a random transfer of momentum between a random pair of random. The difference between these two thermostats is that the former is not momentum-conserving whereas the second one is.

`pbc: [xyz —no ]`

The default is `xyz` and works precisely as in Gromacs. If `no` is chosen, there is a fundamental difference to the way that Gromacs handles it. In DPDMACS, selecting `no` will make the walls impenetrable, thereby allowing to simulate systems with surfaces. At a later stage, this options might be extended to select which surfaces (x-y, x-z or y-z) should be impenetrable.

2. Important parameter entries for readers new to Gromacs

Basically every `.mdp` -file will contain the following entries:

```
integrator    = md
nsteps   = 5000
dt  = 0.001
```

The first line selects that you actually want to run a MD simulation. In the second line, you set the length of the simulation by setting the number of integration step. Each integration step corresponds to the time set by the `dt` parameter, in units of picoseconds, so this example runs a simulation for a total of 5 ps.

The computed trajectory will be written to a `.trr` trajectory file. The trajectory file can contain the coordinates of the particles, their velocities and the forces acting on them. You can select, every how many integration steps these three pieces of data are written:

```
nstxout   = 100
nstvout   = 1000
nstfout   = 0
```

The value `0` will disable the writing. In this example, the coordinates are written every 0.1 ps (assuming `dt  = 0.001`), the velocities are written every 1 ps, and the forces are not written at all.

The simulation can also be thermostated. This means that the velocities of all particles are adjusted in such a way that a canonic ensemble is described. This can be done by the following entries:

```
tcoupl   = andersen
tau_t    = 0.1
ref_t    = 300
```

This selects the Andersen thermostat as means of thermostating. The target temperature is 300 K, and the typical time scale for approaching the target temperature is 0.1 ps.