Overview

This tutorial comes in two parts. In the first, we will look at setting up an implicit solvent simulation on a medium-sized peptide. Later, we will learn how to look at the log file produced by `mdrun`, and learn some of what was going on in our simulation. In the directory for this tutorial, you will find four sub-directories, each with some files you can use to follow along with the tutorial. If you make a mistake, there's backups of input and output for each stage in the `archive` subdirectory for each stage. Don't be lazy though - do type in and run the commands as you proceed through the tutorial to help cement what you learn.

This tutorial assumes you are comfortable using basic UNIX commands, like `cp` to copy files, `cd` to move up and down through the directory hierarchy, and `ls` to see what you have in the current directory. This tutorial is intended for use with GROMACS 4.6.3. You may see minor differences if you use other versions of GROMACS.

The five stages will be looking at

- finding a starting structure
- preparing a topology for a simulation
- energy minimization
- equilibration in implicit solvent
- learning about a big simulation from the log file

Stage 1

Getting a starting structure

Resources like the Protein Data Bank (PDB) are often the starting point for structures for molecular dynamics simulations. In this tutorial, we will look at a structure of three repeats from the tandem octapeptide repeat region within the disordered region of human prion protein. This structure has a PDB entry named 1OEI. This region has five duplicates of the peptide sequence HGGGWGQP; the duplicates are highly conserved among eutherial mammals. So the repeat region is thought to have significant function, but there is a wealth of suggestive experimental data for a large range of different behaviours. A few NMR studies have attempted to shed light on its behaviour, and we will start from a structure from one such NMR ensemble to try to complement this knowledge.

First, we can download such a file, either via a web browser, or on the command line such as with `wget http://www.rcsb.org/pdb/files/1OEI.pdb.gz`. That file is compressed with the GNU compression utility `gzip`, so we can uncompress it with `gunzip` via

```
gunzip 1OEI.pdb.gz
```

You can have a look at the contents of the file later. The experimentalists included twenty models that satisfied the NMR restraints - we'll just use the first one. That model's structure looks like this

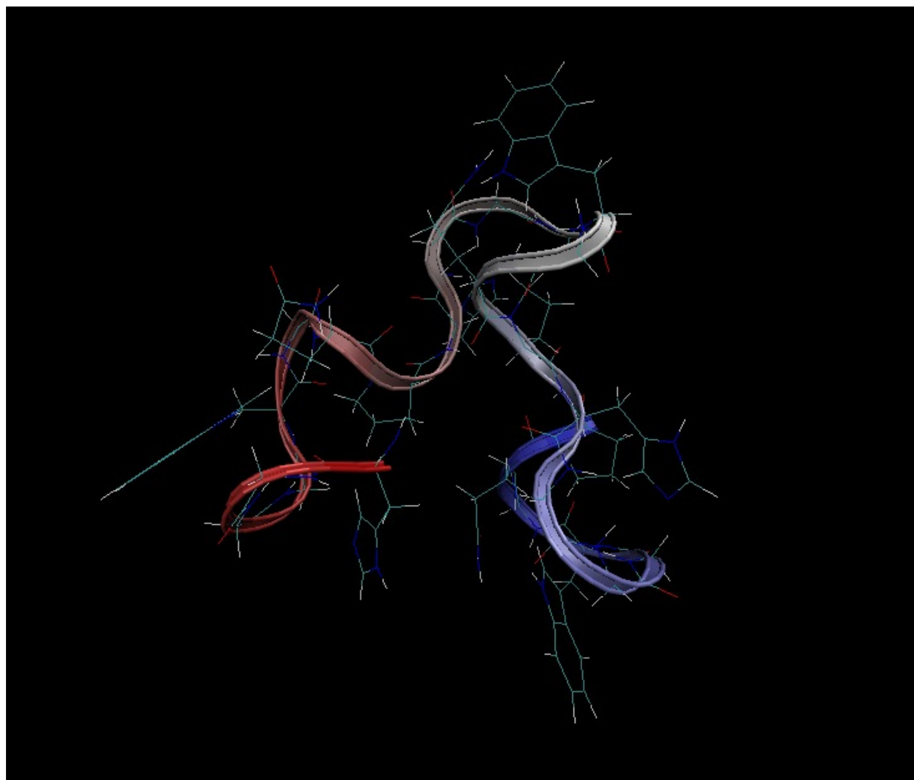

Figure 1: First model of 1OEI PDB entry.

Stage 2

Building a topology

First, we need to set GROMACS up so that we can use it. This normally means you should issue a command such as

```
source /usr/local/gromacs/bin/GMXRC
```

or similar, depending where your copy of GROMACS has been installed. This sets up your environment so that the many parts of GROMACS can work together seamlessly. You will need to do this for each new terminal shell you open (or set up your shell login scripts to do it automatically). Now we can start teaching GROMACS about this peptide with

```
pdb2gmx -f 1OEI.pdb
```

You will be prompted to choose a force field. The different force fields provide different models of real physics that are often different in their ability to reproduce different observables. Choosing a suitable force field is one of the most important decisions you need to make. To make a good decision, you should read the literature for how people have simulated systems similar to yours, and see how good their results were. Then read up on how that force field was derived, and what people have said about it since then. If your choice of model physics is random, so will your results be! Here, we will choose CHARMM27, because that is a good one for small peptides. So type the number next to CHARMM27 and press return.

Now you will be prompted to choose a water model. It is usually a good idea to choose the water model with which the force field was parameterized, or one with which it has been shown to work well for your kind of system. Today, we will choose TIP3P; type its number and press return. Oops, that led to an error. Read on to see how we will solve it!

Working with pdb2gmx to generate a topology

When you run `pdb2gmx`, it looks at the contents of the PDB file, read the GROMACS database for the force field you chose, and tries to match what it knows about that force field with the coordinate file you've supplied. This can be tricky, because there are lots of different conventions for things like atom and residue names. Maybe you need to specify which flavour of the residues you want so you can model a particular pH. You may need to specify whether you'd like charged or neutral termini, or whether disulfide bridges should be made. `pdb2gmx` tries to be smart and do sensible things for you by default, but you should be sure to review what choices it made and decide for yourself whether that will lead to meaningful science!

Here, we see that atom `HB3` in one of the histidine residues wasn't found in the `.rtp` database. There's lots of information and examples in chapter 5 of the GROMACS manual, and that should be your first port of call for figuring out what might have gone wrong. The `.rtp` database is for Residue ToPologies, and it expresses how the force field describes each of the monomers it knows about. `pdb2gmx` has looked at the coordinate file and worked out that residue number 61 is a histidine. Unfortunately, the database expects that the beta carbon is named `CB`, and that its two hydrogen atoms have names `HB1` and `HB2`.

```
[ HSD ]
 [ atoms ]
        N       NH1     -0.47   0
        HN      H       0.31    1
        CA      CT1     0.07    2
        HA      HB      0.09    3
        CB      CT2     -0.09   4
        HB1     HA      0.09    5
        HB2     HA      0.09    6
        ND1     NR1     -0.36   7
```

However, if we look at the coordinate file we supplied, we see

```
ATOM      1  N   HIS A  61        1.671  -1.433  36.155  1.00  0.00           N
ATOM      2  CA  HIS A  61        3.109  -1.661  36.059  1.00  0.00           C
ATOM      3  C   HIS A  61        3.816  -1.746  37.417  1.00  0.00           C
ATOM      4  O   HIS A  61        4.860  -1.146  37.660  1.00  0.00           O
ATOM      5  CB  HIS A  61        3.752  -0.674  35.082  1.00  0.00           C
ATOM      6  CG  HIS A  61        5.068  -1.140  34.503  1.00  0.00           C
ATOM      7  ND1 HIS A  61        5.657  -0.635  33.365  1.00  0.00           N
ATOM      8  CD2 HIS A  61        5.848  -2.180  34.940  1.00  0.00           C
ATOM      9  CE1 HIS A  61        6.769  -1.347  33.126  1.00  0.00           C
ATOM     10  NE2 HIS A  61        6.919  -2.301  34.056  1.00  0.00           N
ATOM     11  H   HIS A  61        1.071  -1.993  35.561  1.00  0.00           H
ATOM     12  HA  HIS A  61        3.217  -2.644  35.613  1.00  0.00           H
ATOM     13  HB2 HIS A  61        3.070  -0.529  34.248  1.00  0.00           H
ATOM     14  HB3 HIS A  61        3.887   0.279  35.593  1.00  0.00           H
ATOM     15  HD1 HIS A  61        5.306   0.130  32.788  1.00  0.00           H
ATOM     16  HD2 HIS A  61        5.645  -2.843  35.776  1.00  0.00           H
ATOM     17  HE1 HIS A  61        7.447  -1.184  32.299  1.00  0.00           H
```

`pdb2gmx` can deal with the atoms being in a different order, but it doesn't know whether the fact that the beta-carbon hydrogen atoms are named `HB2` and `HB3` is a problem or not.

Helping pdb2gmx with atom names

One brute-force solution is to tell `pdb2gmx` to just ignore all the hydrogens in the input file with `pdb2gmx -ignh` and instead re-build them later based on standard geometries. That's often a decent choice, particularly for X-ray crystal structures that were only guessing about the hydrogen atoms, anyway.

A more delicate solution is to go and rename the atoms in the coordinate file to follow the conventions of the residue database, so that `pdb2gmx` can agree with you about what is going on. You could fire up a text editor to do this, but

you need to be really careful to make sure the edits you make don't break the fixed-column format used for many of the kinds of files that describe coordinates. (This made sense in the dark ages, and we're stuck with it now.) You should also take care to use an editor that writes plain text, and if you dare edit on a Windows machine, be aware you will need to use the standard `dos2unix` tool to fix your line endings!

A more powerful way to handle this is to use a UNIX command-line tool. This is particularly useful for writing a script to record how you prepared this topology so you can keep a proper laboratory record! There are many ways to do this, but here we will use the tool `sed` with

```
sed -e 's/HB3 HIS/HB1 HIS/' 1OEI.pdb > 1OEI-fixed-HB3.pdb
```

This uses an *extended regular expression* to transform the name of all `HB3` atoms in histidine residues to `HB1`, and redirect the resulting output to a new file. To check what we changed, we can use another standard UNIX tool called `diff` with

```
diff 1OEI.pdb 1OEI-fixed-HB3.pdb
```

The output format takes a little getting used to, but this is an extremely powerful way to see what is different between two files. Here we can see that the `sed` command has renamed all of the HB3 atoms as HB1, and kept the fixed-column formatting intact. Great, lets go back and try `pdb2gmx` again

```
pdb2gmx -f 1OEI-fixed-HB3.pdb
```

That's irritating, `pdb2gmx` makes us choose the same things each time. Use Control-C to abort, and let's be smarter this time around

```
pdb2gmx -f 1OEI-fixed-HB3.pdb -ff charmm27 -water tip3p
```

Now we're using more of the command line options of `pdb2gmx` to give our instructions up front. Hmm, `pdb2gmx` is unsure about some glycine atom naming, too. This looks like it's going to keep happening for all sorts of carbon atoms in all the residues. Time's ticking, let's just use the big gun

```
pdb2gmx -f 1OEI.pdb -ff charmm27 -water tip3p -ignh
```

Great! That worked. We told `pdb2gmx` to ignore all the hydrogens in the input file, and to build them back afterwards with the proper names.

Checking that a successful pdb2gmx did what you intended

You can see that there's lots of diagnostic output about what `pdb2gmx` did. If you run the `ls` command, you can see that we wrote output files `conf.gro` (which contains the sanitized coordinates of the first model - check it out with `less conf.gro`), `posre.itp` (which might be useful later if we want to use POSition REstraints), and `topol.top`. This last file is crucial - it contains all the knowledge that `pdb2gmx` was able to extract from the database about how this peptide works.

Now, make sure you have a look at the information and make sure there's sensible things there. Should your peptide have a charge? Is the total mass about right? Is the number of chains right? Is the number of residues right? Do the termini make sense? There are lots of options available for `pdb2gmx`, and you can learn about what tools it has through `pdb2gmx -h`.

This peptide is part of a much longer chain, and so far we have modelled it with charged termini. That's wrong if the reason we're doing the simulation cares about the long-chain context. If we're trying to validate the NMR ensemble, we'd better go and check what the termini were there. In fact, there were other residues on either end of this peptide in the NMR experiment, so we probably won't be able to compare our results with theirs unless we do that... but for this tutorial we'll accept the charged termini defaults. If you want, try the `-ter` option of `pdb2gmx` and see how it works.

Also, note that `pdb2gmx` looked at the hydrogen-bonding possiblities and decided that in the sequence HGGGWGQP-HGGGWGQP-HGGGWGQP that the three histidine residues should not all have the same protonation state.

```
Analysing hydrogen-bonding network for automated assignment of histidine
 protonation. 36 donors and 27 acceptors were found.
There are 39 hydrogen bonds
Will use HISD for residue 61
Will use HISE for residue 69
Will use HISE for residue 77
```

This peptide is expected to be pretty disordered, so that deduction from the hydrogen-bonding possibilities in a single structure seems too committal. The pKa values for the two imidazole nitrogens are pretty close (and at pH 7 you get appreciable amounts of the protonated form)... let's just use three copies of the tautomer with the hydrogen on the delta nitrogen. So use

```
pdb2gmx -f 1OEI.pdb -ff charmm27 -water tip3p -ignh -his
```

and choose the first option three times.

Reviewing a pdb2gmx topology in a .top file

Let's take a peek at our topology now. At the top we see something like

```
;
;       File 'topol.top' was generated
;       By user: mark (1302)
;       On host: tcbl02
;       At date: Fri Aug 30 17:30:15 2013
;
;       This is a standalone topology file
;
;       It was generated using program:
;       pdb2gmx - VERSION 4.6.3
;
;       Command line was:
;       pdb2gmx -f 1OEI.pdb -ff charmm27 -water tip3p -ignh -his
;
;       Force field was read from the standard Gromacs share directory.
;

; Include forcefield parameters
#include "charmm27.ff/forcefield.itp"

[ moleculetype ]
; Name            nrexcl
Protein_chain_A     3
```

This repeats back to us details about our command-line method, which is great when you want to remember later on how this all worked. We used GROMACS 4.6.3, with this command line. We're using the `#include` mechanism to do a cut and paste job on the CHARMM27 force field files. This means that any program looking at this file needs to go and get all of that information right now. Then we have a `[ moleculetype ]` declaration for our peptide. Then

```
[ atoms ]
;   nr       type  resnr residue  atom   cgnr     charge       mass  typeB    chargeB      n
; residue  61 HIS rtp HSD  q +1.0
     1        NH3     61    HIS      N      1       -0.3     14.007   ; qtot -0.3
     2         HC     61    HIS     H1      2       0.33      1.008   ; qtot 0.03
     3         HC     61    HIS     H2      3       0.33      1.008   ; qtot 0.36
     4         HC     61    HIS     H3      4       0.33      1.008   ; qtot 0.69
     5        CT1     61    HIS     CA      5       0.21     12.011   ; qtot 0.9
     6         HB     61    HIS     HA      6        0.1      1.008   ; qtot 1
     7        CT2     61    HIS     CB      7      -0.09     12.011   ; qtot 0.91
```

```
  8          HA      61    HIS    HB1      8       0.09      1.008   ; qtot 1
  9          HA      61    HIS    HB2      9       0.09      1.008   ; qtot 1.09
 10         NR1      61    HIS    ND1     10      -0.36     14.007   ; qtot 0.73
 11           H      61    HIS    HD1     11       0.32      1.008   ; qtot 1.05
 ...
```

Here's part of what `pdb2gmx` has been working so hard to achieve from the rtp database. Later, we see

```
[ bonds ]
;  ai    aj funct            c0            c1            c2            c3
    1     2     1
    1     3     1
    1     4     1
    1     5     1
    5     6     1
    5     7     1
    5    18     1
    7     8     1
    7     9     1
    7    12     1
    ...
```

These numbers mean that (say) atoms 1 and 2 have a bond of function type 1. Atoms 1 and 2 are those numbered 1 and 2 in the `[ atoms ]` field above, which here are a nitrogen and hydrogen atom in the N-terminal nitronium group. If there were any special parameters to use when computing how strong those bonded interactions were, they might be listed next on each line.

Down the bottom of the file, we see

```
; Include Position restraint file
#ifdef POSRES
#include "posre.itp"
#endif
```

This lets the topology be smart about whether we might want to use some artificial restraints while we're getting the simulation equilibrated. We'd want to turn those off once things are set up, and the `#ifdef` mechanism lets us do that without editing this file. You can learn more about that if you Google for the so-called *C Preprocessor*. This lets us include the contents of the `posre.itp` file only when we want to. Inside that file, we can see

```
[ position_restraints ]
; atom  type      fx      fy      fz
```

```
 1      1  1000  1000  1000
 5      1  1000  1000  1000
 7      1  1000  1000  1000
10      1  1000  1000  1000
12      1  1000  1000  1000
...
```

This says that our heavy atoms (numbered 1, 5, 7, 10, 12, etc.) might get restrained with restraint type 1 with strength of 1000. Back out in `topol.top` we next see

```
; Include water topology
#include "charmm27.ff/tip3p.itp"

#ifdef POSRES_WATER
; Position restraint for each water oxygen
[ position_restraints ]
;  i funct       fcx        fcy        fcz
   1    1       1000       1000       1000
#endif
```

Next comes another `#include`, this time so that we can re-use the standard water molecule topology. If we were to take a quick peek inside, we see that there's another `[ moleculetype ]` inside it. This is a key point. The topology is built up from a number of molecules, and each molecule might have its own bonds, angles, position restraints, etc. The way this is kept organized is that as soon as a new `[ molecule_type ]` starts, any previous one is finished. If you were to move the `[ position_restraints ]` of one of these molecule types somewhere else (e.g. by putting the `#include` in a random place), then you'll probably get errors complaining about wrong atom indices. Finally, we see

```
[ system ]
; Name
MAJOR PRION PROTEIN

[ molecules ]
; Compound        #mols
Protein_chain_A     1
```

Here the whole system that is present in `conf.gro` is described. It has a name, and it has molecules - well, actually just one molecule. If we wanted to add solvent, then we'd do that later.

Stage 3

Running an energy minimization

Often the starting structure is a bit wrong. Maybe the NMR or X-ray structure had some error. Maybe `pdb2gmx` built some atoms that don't make sense in the complete context. So the first thing we always do is run an energy minimization to relax any really obvious problems. To do that, we have to use the GROMACS tool `grompp`, which takes a pre-built topology, a matching set of coordinates, and combines it with an `.mdp` file which describes the Molecular Dynamics Parameters. `grompp` does lots of error checking at this stage, and in general you can spend a bit of time at this stage. Pay attention to all the output, read the advisory notes, and make sure you make a proper decision about any warnings. You can tell `grompp` to ignore warnings, but you should only do so if you can say why your model physics is appropriate!

Here, our `em.mdp` file will be very simple

```
integrator = steep
nsteps = 25
pbc = no
emtol = 100
```

This uses the steepest-descent energy-minimization algorithm for 25 steps, with no periodic boundaries and a given tolerance on the change in energy. Now let's use the `.mdp` file:

```
grompp -f em
```

This command uses the new file, and picks up by default all the other files we made with `pdb2gmx`, like `conf.gro` and `topol.top`. If you wanted, you could find out from `pdb2gmx -h` which options would allow to specify your output file names if you wanted to make them describe your system, but then you'd have to do the same to tell `grompp` what input files it would need. Your call! This file produces a `.tpr` file (sorry no idea what that stands for!) that contains all the information GROMACS needs to run a simulation. You can even make this file on your laptop and transfer it to your friendly neighbourhood supercomputer and it will always work. Don't try to read it, though, it's in a binary format!

Let's go and use the `.tpr` file we've made:

```
mdrun -v
```

This runs the workhorse of GROMACS that actually does a simulation. We've also asked it with `-v` to produce verbose output. `mdrun` prints some output about what it's doing, and then goes ahead and does it.

```
Steepest Descents:
   Tolerance (Fmax)   =  1.00000e+02
   Number of steps    =           25
Step=    0, Dmax= 1.0e-02 nm, Epot=  1.84360e+03 Fmax= 6.99487e+03, atom= 301
Step=    1, Dmax= 1.0e-02 nm, Epot=  6.64040e+02 Fmax= 3.74784e+03, atom= 301
Step=    3, Dmax= 6.0e-03 nm, Epot=  4.79941e+02 Fmax= 2.89350e+03, atom= 251
Step=    4, Dmax= 7.2e-03 nm, Epot=  3.86274e+02 Fmax= 5.78284e+03, atom= 251
Step=    6, Dmax= 4.3e-03 nm, Epot=  3.36648e+02 Fmax= 6.91968e+02, atom= 301
Step=    7, Dmax= 5.2e-03 nm, Epot=  3.21507e+02 Fmax= 3.87776e+03, atom= 301
Step=    8, Dmax= 6.2e-03 nm, Epot=  2.95247e+02 Fmax= 3.67896e+03, atom= 253
Step=   10, Dmax= 3.7e-03 nm, Epot=  1.93170e+02 Fmax= 1.52563e+03, atom= 253
Step=   13, Dmax= 1.1e-03 nm, Epot=  1.68812e+02 Fmax= 4.42408e+02, atom= 423
Step=   14, Dmax= 1.3e-03 nm, Epot=  1.64730e+02 Fmax= 4.38409e+02, atom= 42
Step=   18, Dmax= 2.0e-04 nm, Epot=  1.62670e+02 Fmax= 4.37372e+02, atom= 421
Step=   21, Dmax= 6.0e-05 nm, Epot=  1.62107e+02 Fmax= 4.37056e+02, atom= 42
Step=   25, Dmax= 9.1e-06 nm, Epot=  1.62023e+02 Fmax= 4.37011e+02, atom= 42

Energy minimization reached the maximum numberof steps before the forces
reached the requestedprecision Fmax < 100.

writing lowest energy coordinates.

Steepest Descents did not converge to Fmax < 100 in 26 steps.
Potential Energy  =  1.6202295e+02
Maximum force     =  4.3701099e+02 on atom 42
    Norm of force     =  1.1506155e+02
```

Here we can see through the behaviour of Epot and Fmax that some energy minimization is taking place. Great! A number of output files were written, which might be of interest for following the progress of the minimization, but which we will not consider now. In particular, the final cooordinates are written out in `confout.gro`, which you would normally check in a visualization program to see that things look sensible.

Stage 4

Equilibration in implicit solvent

Here we will use a new `.mdp` file, and combine it with the output configuration from the energy minimization we just performed. As usual, those files can be found in the archive folder for stage 3. This time we are going to equilibrate our peptide using implicit solvent at 298K, so we need a new `.mdp` file which you can see in `equil.mdp`.

```
integrator = sd
ld-seed = -1
define = -DPOSRE
implicit-solvent = GBSA
pbc = no
comm-mode = angular
rvdw = 0
rcoulomb = 0
rlist = 0
rgbradii = 0
nstlist = 0
constraints = all-bonds
constraint-algorithm = lincs
lincs-order = 4
lincs-iter = 1
nsteps = 1000
nstenergy = 10
dt = 0.002
tc-grps = System
ref-t = 298
tau-t = 1.09
gen-vel = yes
```

We're using stochastic dynamics, with position restraints on the heavy atoms of our peptide, the GBSA implicit solvent method, and some special sauce that means every atom interacts with every other atom. You wouldn't do this for explicit solvent, because it would make the calculation too slow, but with implicit solvent it can actually allow the code to run faster!

This time we need to tell `grompp` the name of the input file, since the default won't work. So

```
grompp -f equil -c confout.gro
mdrun -v
```

will run our equilibration. Lets check how good it was. `g_energy` is a tool that will do a bit of analysis and output on the energy (`.edr`) file that we wrote out every 100 steps above. It contains things other than energies, by the way! We'd like to see a graph, which you can do automatically if you have the Grace graphing tool installed, by using the `-w` flag

```
g_energy -w
```

You can either type in the name of the field you want to see (e.g. Temperature) or the number next to it. Then return, and another return for a blank line to finish. Soon this graph will pop up

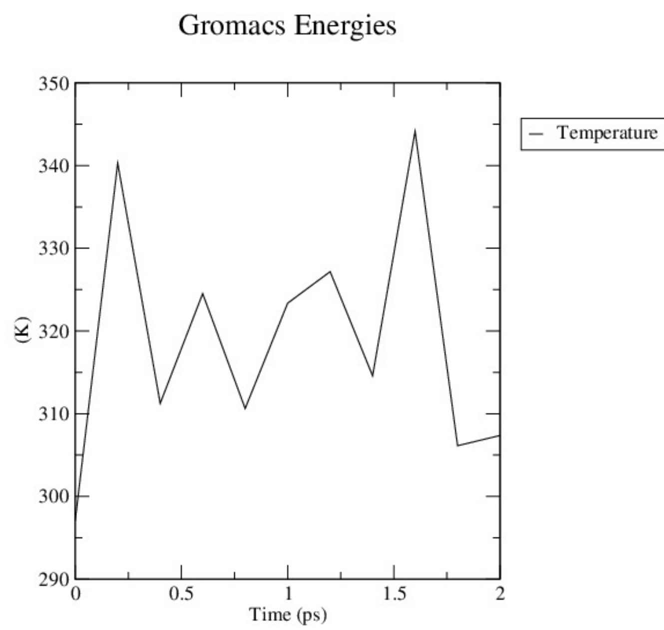

Figure 2: Temperature behaviour over 2ps.

That looks pretty messy - we only equilibrated for 2ps, which is pretty miniscule. In `longer.mdp` there's a much longer equilibration (also with the position restraints turned off, and some new settings that make sure `mdrun` does a continuation of the old simulation, rather than generating new velocities!). You can check out the differences using `diff`, of course

```
diff equil.mdp longer.mdp
```

We also need to tell `grompp` to use the state from the previous simulation, which we do with `-t`.

```
grompp -f longer -c confout.gro -t state -o longer
```

You might not be able to run this simulation in real time for this tutorial, but the resulting energy file is already there as `longer.edr`. If you'd like to try, use

```
mdrun -v -deffnm longer
```

This runs `mdrun`, and tells it that the DEFault FileNaMe prefix for all the input and output files will be `longer`.

We know the start of the equilibration process is probably not representative of the equilbrium, so this time we'll tell `g_energy` to only look at the statistics for the second half of the run. Typing in "Temperature" each time is pretty irritating too. Fortunately, there's several ways to give UNIX tools "canned input," which we'll do via

```
echo Temperature | g_energy -w -f longer -b 100
```

You can see on the terminal that the average temperature was still only about 294K, and that the error estimate is much smaller than the RMSD. So, we know we still haven't got anywhere near enough statistics to be sure we've measured temperature accurately enough to know we've equilibrated well enough. It doesn't seem too bad, though.

That concludes our introduction to preparing topologies and running implicit solvation simulations. When you come to try your hand at these simulations yourself, you will run into problems. The error messages will usually suggest you consult the tips for helping to resolve errors while using GROMACS, and you should be sure to check that page, and look things up in the manual, or on Google, before seeking help on a mailing list.

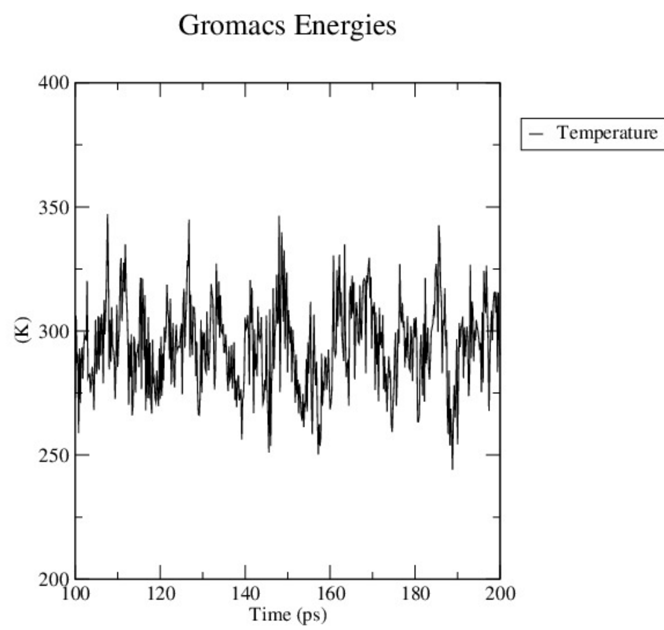

Figure 3: Temperature behaviour over 100ps, after 100ps of equilibration.

Stage 5

What's in a log file?

It's time to knuckle down and learn about the logfile output we get. There's lots of different ways you can run any given simulation, and knowing how to find out whether GROMACS is doing what you asked it means knowing how to read what it reports. You will find that your simulations look different from the example below, particularly if you are only running on a single node, or if you are using accelerators like a GPU.

What simulation are we looking at?

The simulation we'll look at here is totally different from the ones we've just been preparing. It's a 145,000-atom simulation of the Glic ion channel, embedded in a membrane. It was run using MPI parallelism on 30 nodes of Sweden's triolith supercomputer, which has many hundreds of Intel "Sandy Bridge" CPUs, connected by an Infiniband switched network. It's fairly typical of machines you might find available at computing resource centers. GROMACS can run in parallel over many of these CPUs, providing you with the ability to run your simulations faster than you could do on a single CPU.

What does mdrun tell us in the log file?

The output of `mdrun` includes several kinds of information,

- reporting what this `mdrun` can do,
- reporting what simulation is described by the `.tpr`,
- reporting how `mdrun` has decided to implement the simulation of the `.tpr`, now that it knows what the hardware looks like,
- logging some progress results, and
- at the end, reporting on how and where time got spent.

We'll take quick look at each.

Checking mdrun is what you think it is!

Use the UNIX command

```
less triolith.log
```

to take an initial look. Right at the top, we see

```
Log file opened on Sun Aug 25 22:35:31 2013
Host: n39  pid: 18394  nodeid: 0  nnodes:  240
Gromacs version:    VERSION 4.6.3
Precision:          single
Memory model:       64 bit
MPI library:        MPI
OpenMP support:     enabled
GPU support:        disabled
invsqrt routine:    gmx_software_invsqrt(x)
CPU acceleration:   AVX_256
FFT library:        fftw-3.3.3-sse2-avx
Large file support: enabled
RDTSCP usage:       enabled
Built on:           Sun Aug 25 21:47:55 CEST 2013
Built by:           x_marab@triolith1 [CMAKE]
Build OS/arch:      Linux 2.6.32-358.6.2.el6.x86_64 x86_64
Build CPU vendor:   GenuineIntel
Build CPU brand:    Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz
Build CPU family:   6   Model: 45   Stepping: 7
Build CPU features: aes apic avx clfsh cmov cx8 cx16 htt lahf_lm mmx msr nonstop_tsc pcid pc
C compiler:         /software/intel/impi/4.0.3.008/bin64/mpicc GNU gcc.orig (GCC) 4.7.2
C compiler flags:   -mavx    -Wextra -Wno-missing-field-initializers -Wno-sign-compare -Wall
```

The copy of GROMACS that you're using will have many differences here, but there's a number of useful things to check that things are good to use. Important things to verify are

- the GROMACS version - is it what you think it should have been?
- the MPI library - if you're trying to run on a compute cluster, you won't want thread_mpi
- FFT library - should definitely be FFTW or MKL (but if you want maximum performance, there's a few knobs you might want to tweak here)
- single precision - unless you know you need it (e.g. for normal-mode analysis), there's no gain in accuracy from double precision
- CPU acceleration - should suit the hardware you are running on
- C compiler - to get maximum performance, you want the most recent you can possibly get (so generally not the compiler that shipped with your operating system!)
- C compiler flags - should include at least `-O3`

You can get all this output with `mdrun -version`, too.

More performance-critical things reported in the log file

The next section of the log file is mostly highlights of the `.tpr` contents and reminders to you to cite the hard-working theoreticians who worked out the algorithms, and the programmers who wrote the fast, free code you're using! Their salaries depend on it!

After that, there is a mixture of reporting on what `mdrun` has been able to work out about your hardware, and how it's going to implement the simulation described by your `.tpr`. The code for these parts is heavily intertwined, so you'll just have to deal with the changes in context. Of most interest now is the part that reads

```
Using 240 MPI processes
Using 2 OpenMP threads per MPI process
```

This confirms that we're using MPI to distribute the work to 240 processes, and that each process is using two OpenMP threads to share the workload to two cores each. To make sense of this, we have to talk a bit about hardware. These days, all CPUs contain individual computing units called "cores" and these are the fundamental working units. GROMACS tries to allocate the available work to cores in a way that keeps all of them equally busy. However, just like humans in a meeting, if you're all waiting for the last person to show up, or the workload for each person in the team is different, then everybody else's time will get wasted. Humans always have something else to go on with, but dedicated cores on a node of a super-computer are yours to use or waste! Also like humans, there's a limit to how many people can usefully contribute to a project. There's no point convening a meeting of 100 people to get an essay of 1000 words written. You'll spend way longer talking about what to write than actually writing words! It's much better to write in a small group that can have a tight communication loop. How do you find out how many nodes to use on a simulation? That's why we're here!

MPI and OpenMP are the names of two schemes GROMACS uses to split up the workload in a way that is balanced and efficient. Earlier, we saw that 30 nodes of triolith were being used for this simulation, so there are 8 MPI processes on each node. How sensible this is depends a lot on your hardware - so make sure you talk to your friendly system administrators, *after* you read all the documentation they've made available. The nodes of triolith have 16 cores, so 8 groups of 2 cores sounds reasonable. (There's a lot more to "reasonable" than meets the eye, but this tutorial is not the time or place for it! If you're concerned, try some alternatives and use what's fastest.)

Next, we see

```
Detecting CPU-specific acceleration.
```

```
Present hardware specification:
Vendor: GenuineIntel
Brand:  Intel(R) Xeon(R) CPU E5-2660 0 @ 2.20GHz
Family:  6  Model: 45  Stepping:  7
Features: aes apic avx clfsh cmov cx8 cx16 htt lahf_lm mmx msr nonstop_tsc pcid pclmuldq pdc
Acceleration most likely to fit this hardware: AVX_256
Acceleration selected at GROMACS compile time: AVX_256
```

`mdrun` is reporting on some nuts and bolts here - the most important thing is that the compile-time and run-time acceleration matches. It sometimes happens that the front end machine of a compute cluster is different (older! slower!) from the real machines, and you want to make sure that `mdrun` was compiled with the right target in mind. Otherwise you might see `mdrun` fail mysteriously, or run slower than it should.

How fast did it run?

Let's jump to the end and have a look at the performance results. You can do that with `less` by using the key combination `Shift-g`. The key information is here

```
               Core t (s)   Wall t (s)        (%)
       Time:    13194.330       37.187    35480.8
                 (ns/day)    (hour/ns)
Performance:      232.349        0.103
Finished mdrun on node 0 Sun Aug 25 22:36:46 2013
```

Here we see how long this simulation took and what throughput it achieved. This is the primary way you'd want to measure whether you are using an efficient amount of hardware. Just like humans in meetings, the more hardware you try to use, the more overhead there is from communication. Worse, if your network is a shared resource with other users on the machine, the more hardware you use, the more likely you are to get hit by others' traffic. Talk to your system administrators about how best to get a chunk of the network to yourself!

Scaling

In practice, there's a "linear" regime where doubling the number of cores doubles the performance, and gradually as you start adding more hardware you get less and less value for it. It's up to you to choose whether you want to use your resources efficiently to generate the maximum amount of results over a longer period, or whether you want to get a result as quickly as you can. If you are going to run the same kind of simulation on the same hardware for months, then it is worth your time to look at a few alternatives. It's difficult to give

general advice, because it depends critically on your hardware, your network, your compiler, and the model physics in your `.tpr`; but in GROMACS 4.6.x you should certainly be able to run at significantly under 1000 atoms/core, but might not achieve linear scaling while doing it.

This simulation had 145,000 atoms across 480 cores on 30 nodes, which is only 300 atoms/core. If we were to look at the `triolith-24-nodes.log` file for the same calculation on only 24 nodes, we see 215 ns/day. Adding 25% more hardware only added 10% more performance. If you needed to run a microsecond this week to make a deadline, go with 30 nodes, but don't do it blindly!

This brings us to another important practical point in running MD simulations. It's not necessarily a good thing to be able to use a lot of hardware to run a simulation, if a smaller amount of hardware or a different MD program will produce higher throughput. Doing good science is often about minimising the cost while maximising the length of the simulations, and not about how much hardware you used to do it!

Why didn't it scale?

How can we tell what's running slow and thus why? Let's take a look just above the end of the log file.

```
     R E A L   C Y C L E   A N D   T I M E   A C C O U N T I N G

 Computing:         Nodes   Th.     Count  Wall t (s)     G-Cycles       %
-----------------------------------------------------------------------------
 Domain decomp.       180    2       2000       4.265     3369.786     8.6
 DD comm. load        180    2       2000       0.022       17.387     0.0
 DD comm. bounds      180    2       2000       0.164      129.453     0.3
 Vsite constr.        180    2      20001       0.647      511.316     1.3
 Send X to PME        180    2      20001       0.042       32.887     0.1
 Neighbor search      180    2       2000       2.824     2231.473     5.7
 Comm. coord.         180    2      18001       1.290     1019.426     2.6
 Force                180    2      20001      12.924    10210.398    26.1
 Wait + Comm. F       180    2      20001       2.901     2291.742     5.9
 PME mesh              60    2      20001      23.067     6074.587    15.5
 PME wait for PP       60                      14.121     3718.642     9.5
 Wait + Recv. PME F   180    2      20001       6.028     4762.342    12.2
 NB X/F buffer ops.   180    2      56003       0.412      325.177     0.8
 Vsite spread         180    2      22002       0.809      639.312     1.6
 Write traj.          180    2          1       0.007        5.465     0.0
 Update               180    2      20001       0.181      143.296     0.4
 Constraints          180    2      20001       4.176     3299.503     8.4
 Comm. energies       180    2       2001       0.229      181.115     0.5
 Rest                 180                       0.199      209.614     0.5
```

```
--------------------------------------------------------------------------------
 Total                   240                               37.187    39172.919   100.0
--------------------------------------------------------------------------------
```

This table shows the 240 MPI ranks split into 180 PP nodes and 60 PME nodes. If you've got no idea what that's all about, go and check out the GROMACS manual section 3.18.5 "Multiple-Program, Multiple-Data PME parallelization." In particular, the following flowchart from the manual has sections that map pretty closely to the above table entries.

Basically, the non-bonded part of the force calculations dominates the cost. The most efficient algorithm that computes the long-ranged part of that accurately is called PME. It works best if only a small number of "PME" nodes participate in the Fourier-space part, because all of those nodes have to communicate with all of the others.

If we didn't split the nodes into two groups, this job would have run much slower than it did! `mdrun` does a pretty good job of choosing how to manage this PME-PP split by default. `mdrun -tunepme` is on by default, and will tweak some things for you automatically for best performance. If you really want to experiment with getting it right for your system and hardware, have a look at the GROMACS `g_tune_pme` tool.

Above, we can see nearly 10% of the time is spent with the PME nodes waiting to get some work to do. Ideally, we'd set up the `.tpr` file so that we'd have neither set of nodes ever waiting for the other. In practice, that isn't possible because not all parts of the MD algorithm can be usefully parallelised, so the PME nodes have to waste some time. Worse, on a shared network, achieving good PP-PME load balance is pretty much impossible because you never know how much traffic will be on the network, or when. You should be prepared to accept a few percent of waste in "PME wait for PP" because it is inevitable, and at least the number of PME nodes doing that is never larger than the number of PP nodes.

Some red flags in the performance table

There are some red flags above: `Wait + Comm.  F` would be small if all the PP nodes had the same amount of work to do. The 5.9% we see here is pure waste. In practice, balancing the PP load across lots of cores is hard to do if your simulation has any kind of heterogeneity. This system has water (with no bonded interactions), membrane (with few electrostatic interactions), and protein (with everything), and there's no way a *spatial* domain decomposition can give every PP node an equal slice of the action and not drown in communication overhead. So, `mdrun` starts with a regular spatial grid of domains on the PP nodes, and over time moves the boundaries so that the domains that (say) deal with protein have smaller volumes than the domains that are just doing water molecules.

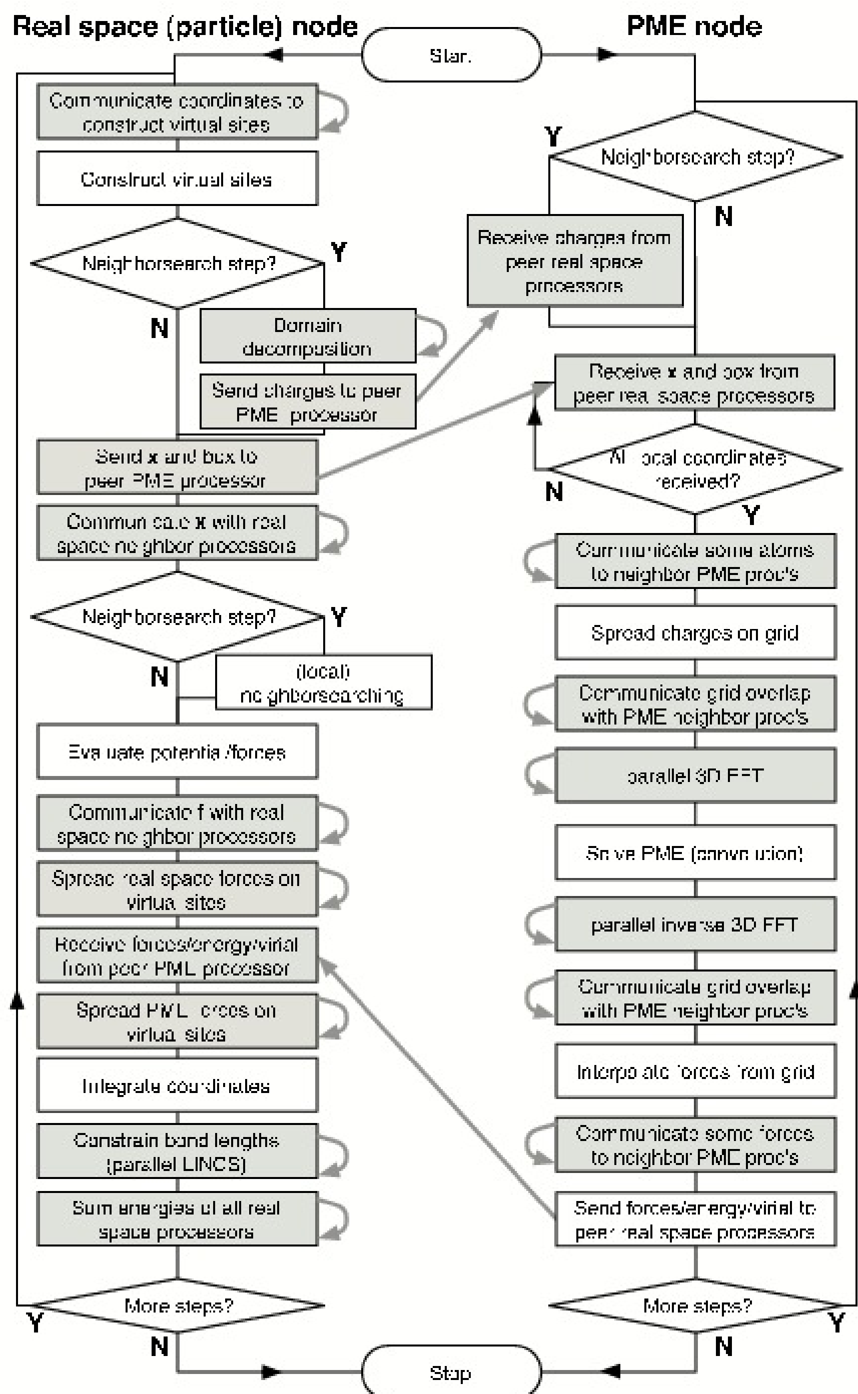

Figure 4: Flow chart showing the algorithms and communication (arrows) for a standard MD simulation with virtual sites, constraints and separate PME-mesh nodes.

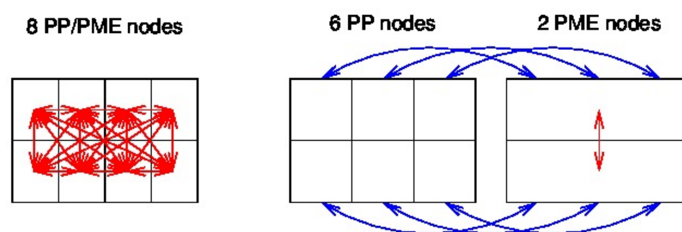

Figure 5: Example of 8 nodes without (left) and with (right) MPMD. The PME communication (red arrows) is much higher on the left than on the right. For MPMD additional PP - PME coordinate and force communication (blue arrows) is required, but the total communication complexity is lower.

How well this works in practice depends on a lot of things. There has to be a minimum volume for each domain, which depends on how your `.mdp` file set up constraints and non-bonded cut-offs. More on this later.

This means that `mdrun` is inefficient at the start of the simulation. The DD is regular at the start, which will have unbalanced load. This gets better over time. How important this is depends on the heterogeneity of your simulation, but you will get more reliable performance numbers if you tell `mdrun` to only collect performance statistics over the later part of the run. `mdrun -resetstep` or `mdrun -resethalfway` are your friends, there. A thousand or so MD steps is normally good enough for DD to get properly warmed up, but your mileage will vary. If your network is noisy, you should do a several short runs to see what sort of variation you might get.

Another red flag is `Wait + Recv.  PME F` at 12.2%, because not only were the PME nodes waiting to get work to do, but then the PP nodes had to wait to get the results from the PME nodes. Yuck! Ideally, the PP nodes would finish work and find the results from the PME nodes all ready to go. If we look a little higher in the log file we'll see that `mdrun` tells us when conditions are bad enough that we probably want to do something about it:

```
 Average PME mesh/force load: 1.386
 Part of the total run time spent waiting due to PP/PME imbalance: 14.0 %

NOTE: 14.0 % performance was lost because the PME nodes
      had more work to do than the PP nodes.
      You might want to increase the number of PME nodes
      or increase the cut-off and the grid spacing.
```

Adding more PME nodes would help if the simulation was being limited by the amount of work the PME nodes had to do. How can we tell? Below the first table we looked at, we can see

```
-----------------------------------------------------------------------------
 PME redist. X/F       60     2       40002         4.370      1150.929     2.9
 PME spread/gather     60     2       40002         7.372      1941.312     5.0
 PME 3D-FFT            60     2       40002         3.120       821.543     2.1
 PME 3D-FFT Comm.      60     2       80004         7.545      1987.038     5.1
 PME solve             60     2       20001         0.616       162.117     0.4
-----------------------------------------------------------------------------
```

This breaks down the 15.5% of the run time that was reported in `PME mesh`. One third of that was already spent doing the 3D-FFT communication, and there's other communication going on as well, so it doesn't sound like adding more processes that need to do more communication would be much help! Another option would be to try to change the balance of the number of OpenMP processes to MPI nodes. Whether this is useful varies a lot - try it and see!

Load balance between particle-particle processes

How do we tell if particle-particle (PP) load balance is a problem? Also at the end of the log file we see:

```
 Average load imbalance: 14.3 %
 Part of the total run time spent waiting due to load imbalance: 4.5 %
 Steps where the load balancing was limited by -rdd, -rcon and/or -dds: X 0 % Y 0 % Z 10 %
```

On 10% of the neighbour-search steps, the load-balancing algorithm would have liked to change the sizes of the DD cells, but the way we set up the simulation didn't let it. The heterogeneity in the Z dimension (conventionally, perpendicular to the membrane) was too hard to deal with for this number of cores. In fact, we can see this during the run, too, because when it notices these kinds of problems, `mdrun` prints out lines like

```
DD  step 2510029879  vol min/aver 0.430! load imb.: force 14.9%  pme mesh/force 1.440
```

where we can see that the ratio of the smallest to average DD cell volume is 0.43. That means some domains have volumes that are less than half that of the average, which is pretty extreme. The exclamation point means the load balancing algorithm would like to have done things, but it couldn't do so safely without risking the simulation crashing. Why was that? Let's go back to about line 290 of `triolith.log` (e.g. enter `290 Shift-g` in `less`):

```
Initializing Domain Decomposition on 240 nodes
Dynamic load balancing: yes
Will sort the charge groups at every domain (re)decomposition
Initial maximum inter charge-group distances:
    two-body bonded interactions: 0.430 nm, LJ-14, atoms 9084 9092
  multi-body bonded interactions: 0.430 nm, Proper Dih., atoms 9084 9092
Minimum cell size due to bonded interactions: 0.473 nm
```

Here we know we want DD to work on 240 nodes, and the minimum cell size required for the atoms in bonded interactions to be automatically available when we want them is 0.473 nm. That makes sense - a 1-4 interaction occurs over four bonds that are each in the range of about 0.1-0.15 nm.

```
Maximum distance for 7 constraints, at 120 deg. angles, all-trans: 1.134 nm
Estimated maximum distance required for P-LINCS: 1.134 nm
This distance will limit the DD cell size, you can override this with -rcon
```

Here's something that can be a problem. Using P-LINCS for constraints requires that once the forces are computed, the position update is done in such a way that

the constraints are still true after the positions are updated. However there are lots of atoms participating in multiple bonds, and the constraint correction has to take all of them into consideration, and sometimes those atoms live on different PP nodes. Ugh! In the `.mdp` file for this simulation, I chose `lincs-order = 6`, which means a DD cell will have to tell its neighbours it needs coordinates for atoms 6+1 bonds away. Because it's efficient to let each DD cell only talk to its nearest neighbours, that creates a minimum size for a cell. That was probably why this simulation had troubles with load balancing in the Z dimension. Check out the GROMACS manual section on `lincs-order` and `lincs-iter` for ways you can play with this if you think it is a problem for you. Note that using `constraints = h-bonds` is much less of a problem here. You may get better throughput that way, even if you have to reduce the MD step size!

```
Guess for relative PME load: 0.21
Will use 180 particle-particle and 60 PME only nodes
This is a guess, check the performance at the end of the log file
Using 60 separate PME nodes, as guessed by mdrun
```

Here's `mdrun` reporting on its guesses about how many PME processes to use. `mdrun -npme` will let you specify this if you want to try alternatives. It is very important that there are a lot of factors for both the number of PP and the number of PME processes, and preferably all of 2, 3 and 5! This has to do with being able to do spatial decompositions that will be efficent with lots of the different kinds of communication that has to go on within and between the groups of processes.

```
Scaling the initial minimum size with 1/0.8 (option -dds) = 1.25
```

Here's a "fudge factor" that helps try to keep the initial domains as spherical as possible, to minimize the volume of atoms that have to be communicated.

```
Optimizing the DD grid for 180 cells with a minimum initial size of 1.417 nm
The maximum allowed number of cells is: X 6 Y 6 Z 10
```

The size of your simulation box affects the maximum number of DD cells, now that `mdrun` knows how big each cell should be. Finally, here's the result of the domain decomposition:

```
Domain decomposition grid 6 x 5 x 6, separate PME nodes 60
PME domain decomposition: 6 x 10 x 1
```

Notice how the number of PP and PME cells X dimension is the same, and the Y dimension has a common factor. If the number of nodes `mdrun` has available can't do that kind of thing, then the maximum performance will suffer, because

the communication pattern will be awkward. If you're getting warnings about load balance being bad in a particular dimension, have a look here and consider the number of slabs that dimension is being chopped into. You might want to try specifying other DD grids (see `mdrun -h`) or even rotating your whole system!

If you're using a different kind of network than Infiniband, and you're at about the scaling limit of GROMACS on your kind of simulation, then you might like to investigate how best to lay out your MPI processes so that those that talk the most are neighbours.

Conclusion

So there's our overview of how to find useful information in a log file. Most of it is not important until you are running lots of simulations for a long time and need to know how best to juggle settings to make best use of finite resources. Don't spend too much of *your* time on it though. The computer is cheaper than you are, even if your salary doesn't make it feel that way!

Author, feedback, and getting help

This tutorial was prepared by Mark Abraham for the 2013 GROMACS USA Workshop. Feedback on the format and content is warmly invited. General MD and GROMACS questions arising from the above discussion should go to the GROMACS users mailing list. I am unable to give private help to individuals as they work their way through this tutorial, nor with general molecular simulation tasks.