File Conversion Among MD Simulation Engines Using ParmEd

ParmEd is a versatile Python library that facilitates the interconversion of files between popular molecular dynamics (MD) simulation engines like Gromacs, Amber, and NAMD (CHARMM). This tool is especially useful for researchers and students working in molecular dynamics who need to switch between simulation packages without hassle. For example, you want to avoid setting up a protein-ligand complex in Gromacs (adding ligands to gmx force field files can be troublesome!) but do want to run MD simulations in Gromacs for its speed. You will need to use ParmEd to convert the Amber files to Gromacs format.

Note that the MD engine uses different algorithms and settings. You cannot either adopt special settings in another MD engine (e.g. restraints, you should set it up again). You should not even wish to fully replicate a Gromacs simulation in Amber. But for most biological systems (e.g. the solvent is not that important), MD engine usually affects your simulation much less than other options, like the choice of force field. So feel free to switch between MD engines!

Jump to the code section if you want a solution only.

Installing ParmEd

Here's how you can install ParmEd using Anaconda:

```
conda install -c conda-forge parmed
```

If you have compiled Amber on your system, you might already have ParmEd installed as part of the AmberTools suite. To ensure it is properly integrated, refer to the comprehensive guide on compiling Amber, which is particularly useful if you are setting up everything from scratch.

Introduction

Knowing the file formats

These file formats are what we need in MD simulations:

Engine	Construction Tool	Topology file	Coordinate file	Parameter file
Gromacs	`pdb2gmx`	`.top`/`.itp`	`.gro`	--
Amber	`tleap`	`.prmtop`	`.inpcrd`	--
NAMD	VMD `psfgen`	`.psf`	`.pdb`	`.prm`

ParmEd logics

ParmEd works simply: read in the topology and coordinate files, and write out two files in the desired format.

ParmEd writes the parameters into `.inpcrd` (as it is) and `.top` files. Always find `.prm` files when converting both from and to NAMD.

Other

You can edit the system in ParmEd, which is out of the scope of this post. The file parsing is very detailed so you can manipulate the system as you like. Consult the ParmEd documentation for more details.

Conversion Code

The following code shows a framework of file conversion. It implements the basic residue renumbering function: you can set the starting residue number. The command is

```
python xxx.py <system_name> <starting_residue_number>
```

Your topolgy and coordinate files should be named `<system_name>.xxx` both. Note that we use `offset-1` in the code since by default ParmEd residue numbers start from 1.

> ⚠ **Warning**
>
> Always double check after the conversion!

> ⚠ **Warning**
>
> For a very large system (hundreds of thousands of atoms), this process could take some time.

From Amber to Gromacs

```
# python amber2gmx_via_parmed.py pro 689

import parmed as pmd
import sys

prefix = sys.argv[1]
offset = int(sys.argv[2])
amber = pmd.load_file(prefix+'.prmtop', prefix+'.inpcrd')

# renumbering
for residue in amber.residues:
    _ = residue.idx  # Get the original index
    residue._idx += offset-1
    residue.number += offset-1

# Save the modified files in Gromacs format
amber.save(prefix+'.top', overwrite=True, combine='all')
amber.save(prefix+'.gro', overwrite=True, combine='all')
```

Gromacs sub-topology `.itp` files can be read, but cannot be written, i.e. ParmEd writes huge topology/coordinate files without subfiles as in Amber/NAMD.

From CHARMM to Gromacs

```
# python charmm2gmx_via_parmed.py pro 689

import parmed as pmd
```

```
from parmed.charmm import CharmmParameterSet
import sys
prefix = sys.argv[1]
offset = int(sys.argv[2])

structure = pmd.load_file(prefix+'.psf')
# renumbering
for residue in structure.residues:
    _ = residue.idx
    residue._idx += offset-1
    residue.number += offset-1
parameter = CharmmParameterSet('par_all36m_prot.prm',
'toppar_water_ions_namd.str')  # add more if necessary

# edit the sign of epsilon
for atomname, atomtype in parameter.atom_types.items():
    atomtype.epsilon *= -1
    atomtype.epsilon_14 *= -1
structure.load_parameters(parameter)

# Save the modified files in Gromacs format
structure.save(prefix+'.top', overwrite=True, combine='all')
structure = pmd.load_file(prefix+'.pdb')
structure.save(prefix+'.gro', overwrite=True, combine='all')
```

> **Tip**
>
> ParmEd does not realize that for epsilon gmx adopts the absolute value while charmm files store the real value (negative!)

> **Note**
>
> In parameter files like `par_all36m_prot.prm` downloaded from CHARMM website, officially all atom type definitions are commented, but we should uncomment them for parmed, or it cannot find atomtypes. Or read `.rtf` files too. Double check your files!

From Gromacs to Amber

```
# python gmx2amber.py system

import parmed as pmd
import sys
prefix = sys.argv[1]

parm = pmd.load_file(prefix+'.top', prefix+'.gro')
# Save the modified files
parm.write(prefix+'.prmtop')
parm.write(prefix+'.inpcrd')
```

I actually have not tried this (see problems). You may need to add residue renumbering mechanisms. Practice yourself! And I guess from CHARMM to Gromacs works similarly.

Renumber gmx files

This adopts the similar process. The original files are overwritten.

```
# python gmx_renumber_via_parmed.py pro 689

import parmed as pmd
import sys
prefix = sys.argv[1]
offset = int(sys.argv[2])
gmx = pmd.load_file(prefix+'.top', prefix+'.gro')

# renumbering
for residue in gmx.residues:
    _ = residue.idx
    residue._idx += offset-1
    residue.number += offset-1
# regenerate and revalidate the internal parameters, usually do this after
modifying the structure
gmx.remake_parm()
# Save the modified files
gmx.save(prefix+'.top', overwrite=True)
gmx.save(prefix+'.gro', overwrite=True)
```

From CHARMM to Amber

To convert CHARMM files to Amber format, use chamber:

```
chamber -top topol.rtf -param params.par -str stream.str -psf structure.psf -crd
structure.crd -outparm amber.prmtop -outcrd amber.inpcrd
```

- Topology files (`-top`, `-str`) are only necessary if the parameter files do not define the atom types
- Parameters (`-str`, `-param`) are applied to your structure
- `-crd` option accepts file formats like PDB, CHARMM CRD, Amber restart, etc.

Issues

Residue renumbering

Problem

None of these file formats are perfect.

- Gromacs files do not have chain identifiers. By default chains are separated into a few `.itp` files, so it's hard to locate an atom in a specific chain in a `.gro` file.
- Amber files always start with residue numbers 1, which causes trouble when aligning with the "biological" residue nubmers.
- VMD files have full identifiers. However, we have to manually separate the chains when modeling.

You cannot change the file formats unless your write your own MD engine. So just put up with it...

With ParmEd, you can try to edit the residue numbers to match the "biological" residue numbers. Sadly, if you have multiple chains and they are overlapping, you still have to use that sequential residue numbers. But if you have only one chain, this won't bother you.

Edit in VMD

During visualization in VMD, you can edit the residue numbers like this:

```
mol new system.prmtop type parm7 first 0 last -1 step 1 filebonds 1 autobonds 1
waitfor all
mol addfile md.nc type netcdf first 0 last -1 step 1 filebonds 1 autobonds 1
waitfor all
# select whatever you are interested, but too many water many slow down the
process
set all [atomselect top "protein or resname LIG or resid 1 to 1500"]
foreach idx [$all get index] {
    set atom [atomselect top "index $idx"]
    $atom set resid [expr [$atom get resid] + 688]
}
```

Edit in ParmEd

In ParmEd, every `Residue` object in a `Structure` has an `idx` attribute. This attribute indicates the residue's index within the structure, and it is managed internally by ParmEd. It is crucial not to modify this attribute directly, as it could lead to inconsistent state within the structure. Some other attributes are also private and cannot be modified.

Anyway, I've figured out the code to edit residue numbers. I don't really know why I have to manipulate `_idx`, but it works. Feel free to inspect the attributes when debugging in your IDE, and create your own workflow!

Parameters and atomtypes

GROMACS: Independent Parameter Specification

In GROMACS, topology files (typically `.top`) allow for each bond term to be specified independently. This means that different bond parameters can be assigned to the same pair of atom types, provided they occur in different contexts within the molecule.

Example of a GROMACS bond specification:

```
; Bond parameters
; i    j   func   length  force_const
  1    2    1      0.123   456.7   ; Asymmetric bond A
  2    3    1      0.123   456.7   ; Asymmetric bond B
```

CHARMM: Type-Based Parameter Definition

Conversely, CHARMM typically defines parameters between different atom types based on a consistent set of parameters across all bonds involving those atom types. This approach assumes that identical pairs of atom types will always exhibit the same bonding characteristics, regardless of their molecular environment.

```
BONDS
CA    CB    340.0  1.529   ; Standard peptide bond
CA    CG    317.0  1.510   ; Standard alkane bond
```

Resolving Parameter Inconsistencies

When converting from GROMACS to CHARMM formats using tools like ParmEd, discrepancies in how bond parameters are specified can lead to errors. For instance, ParmEd might encounter a `ParameterError` if it detects different bond parameters for the same atom types, which is permissible in GROMACS but not in CHARMM. This issue is particularly evident with complex ions or molecules optimized asymmetrically through QM methods, such as $\mathrm{Al(OH)(H_2O)_5}^{2+}$.

To address these conversion challenges, users have two main options:

1. **Assign Different Atom Types:** Modify the topology to assign unique atom types for bonds that require different parameters.
2. **Uniform Bond Parameters:** Standardize bond parameters for each pair of atom types, ensuring consistency across the entire molecule.

For more details on handling these conversions and the underlying code structure of ParmEd, consider exploring the following resources:

- ParmEd GitHub repository
- Issue related to parameter mismatches
- Discussion on handling different parameters

End

We welcome your feedback and contributions! If you have developed new workflows or if you encounter any issues, please don't hesitate to reach out. For reporting problems, consider opening an issue on the ParmEd GitHub repository. Your insights and experiences are invaluable in enhancing the tools and community resources.