CECAM Winter School 2023: Tinker

February 9, 2023

Abstract

Hereafter is the material and documentation associated with the CECAM Winter School dedicated to the functionalities and the usage of the Tinker modelling package on the 10th of february 2023. The Tinker package comes with several programs to run molecular simulations, from system preparation to molecular dynamics up to analysis of the results of such simulations. It has a particular focus on the use of advanced polarizable force fields such as AMOEBA whose electrostatic interactions include a polarization term modelling the induction response of atomic electron clouds to the changing environment. This higher resolution description of the molecular interactions comes with a higher computational cost, which is why High Performance implementations such as Tinker-HP (for CPUs and GPUs) and Tinker 9 (for GPUs) have been developed. These implementations share the structures of their inputs and outputs and many of their core functionalities. Still, this document has a particular focus on Tinker-HP.

1 Introduction

The goal of this school is not to review all of the (many) functionalities of the Tinker programs but to focus on the most basic ones such as system preparation, energy analysis, standard molecular dynamics and simple enhanced sampling simulations. For a more complete description of all these possibilities, the reader can look at the Tinker user guide available on Jay Ponder's web page dedicated to Tinker. For the more complex case of the parametrization of a new system with AMOEBA, the reader can look at the poltype2 program that can handle this procedure systematically.

Other tutorials can be found in the tutorial section of Tinker-HP's website, there is for example one about system's preparation.

1.1 Prerequisite

- an environment with access to a GPU or CPUs
- VMD visualization software: VMD
- clone Tinker-HP's github repository: "git clone https://github.com/TinkerTools/tinker-hp.git ."
- get Tinker 8 executables : web page dedicated to Tinker
- clone the "Tinker Winter-School 2023" repository to get the file associated to this course: "git clone https://github.com/louislagardere/winter-school-2023.git ."

1.2 input files

There are essentially three types of input files associated with the Tinker programs, as well a command line specific to each of these executables: ***.xyz**, ***key** and ***prm** files.

1.3 *.xyz files

Tinker uses a specific **xyz** format to represent the topology of the systems, such files are made of:

- a first line with the number of atoms of the systems and a header

- a line for each atoms with separated columns representing:
  - index of the atom
  - tag ("H", "O", "CA"...)
  - x, y and z cartesian coordinates
  - **atom type** of the atom
  - **connectivity**: list of the indexes of the atoms covalently bonded to the current one

A typical example of such a file for a water box is shown below. Note that this file format can be read by the vizualisation software **VMD** as the **Tinker** file format.

```
 12000  Water Cubic Box (49.323 Angstroms, 4000 AMOEBA)
     1  O     -2.722762   15.350993   -0.920138     1     2     3
     2  H     -2.603766   14.476160   -1.273219     2     1
     3  H     -3.543810   15.582853   -1.332255     2     1
     4  O    -10.815448   -4.263334   -8.969854     1     5     6
     5  H    -10.289466   -3.650951   -8.469188     2     4
     6  H    -10.287341   -5.026283   -9.197981     2     4
     7  O     -1.812615   13.230262  -16.468579     1     8     9
     8  H     -1.688057   12.307974  -16.749344     2     7
     9  H     -1.030979   13.732442  -16.805708     2     7
    10  O    -14.390481   23.068070  -20.491394     1    11    12
    11  H    -15.031721   23.768527  -20.680539     2    10
    12  H    -14.918601   22.330280  -20.137210     2    10
    13  O      4.166808  -21.008269   -9.769746     1    14    15
    14  H      3.731726  -21.614069  -10.414392     2    13
    15  H      3.537775  -20.295515   -9.558323     2    13
    16  O    -13.584078  -13.261964   13.179153     1    17    18
    17  H    -14.543373  -13.033032   13.373306     2    16
    18  H    -13.002261  -12.483462   13.276235     2    16
    19  O      0.118442  -22.005784  -13.663408     1    20    21
    20  H      0.587254  -21.167693  -13.314399     2    19
    21  H     -0.837232  -21.932836  -13.451332     2    19
    22  O     -7.693084  -23.898108   -6.755711     1    23    24
    23  H     -7.835075  -24.662297   -7.327124     2    22
    24  H     -8.177253  -23.204571   -7.199697     2    22
    25  O     -3.056125  -14.363931   23.736073     1    26    27
    26  H     -3.134540  -15.225368   23.287860     2    25
    27  H     -2.322650  -14.452610   24.240729     2    25
    28  O     22.688643   -0.899118  -23.031435     1    29    30
    29  H     23.316868   -0.182049  -23.130030     2    28
    30  H     22.008459   -0.638371  -22.348766     2    28
    31  O     -7.291214   -0.941419   -8.299583     1    32    33
    32  H     -6.527084   -1.565220   -8.301774     2    31
    33  H     -7.604609   -1.032992   -9.213869     2    31
    34  O    -17.110989  -22.571827    7.240290     1    35    36
    35  H    -17.230431  -22.797403    6.331462     2    34
    36  H    -16.294192  -21.936606    7.255374     2    34
    37  O     -8.363650  -21.199533   -5.374122     1    38    39
    38  H     -8.902690  -20.677223   -5.988821     2    37
    39  H     -8.959553  -21.428613   -4.648638     2    37
    40  O    -24.484823   -4.497322  -22.489194     1    41    42
    41  H    -25.324308   -3.990816  -22.509595     2    40
    42  H    -24.461732   -4.981904  -23.262751     2    40
    43  O    -17.408174    5.085603  -24.112996     1    44    45
    44  H    -16.866405    5.585437  -23.501880     2    43
    45  H    -17.485793    5.625474  -24.914662     2    43
    46  O    -16.579549    1.017791  -10.209087     1    47    48
    47  H    -16.454056    0.052302  -10.128796     2    46
    48  H    -17.061690    1.088578  -11.012688     2    46
    49  O     11.233506   -8.052452  -21.127540     1    50    51
    50  H     11.349015   -8.778138  -20.480946     2    49
    51  H     10.615703   -8.547637  -21.737627     2    49
    52  O     -6.171174  -17.834971   17.918873     1    53    54
    53  H     -5.654911  -16.990945   18.191071     2    52
    54  H     -6.646119  -18.068860   18.680046     2    52
    55  O     24.590935  -20.009008   13.257527     1    56    57
    56  H     24.261195  -19.940128   12.296944     2    55
"waterhuge.xyz" 12001L, 744056C
```

Figure 1: beginning of a typical xyz file with as many line as atoms in the system and 7 columns representing: index, tag, x,y and z cartesian coordinates, atom types and connectivity for each atom

1.4 *.key files

The *key files associated to a simulation encompasses the keywords that will drive the computation (on top of the default ones). Examples of such keywords are :

- the **parameter** file containing, for each atom type, the list of parameters of the force field uses. These are typically the parameters associated to the valence terms such as ideal bond length and strength or equilibrium bending angles, and non bonded parameters such as partial charges or multipoles, polarizabilities and van der Waals parameters.
- size/shape of the simulation box in periodic boundary conditions
- thermostats, barostats
- integrators for MD...

If no specific *key file is specified in the command line, running for an input **system.xyz**, the programs will look for a key file with the same prefix: **system.key**. The *key file can be specified by adding the following to the command line **-k otherkeyfile.key**

```
parameters        water03
verbose

a-axis                        49.232

thermostat                  BUSSI

vdw-cutoff                     12.0

ewald
ewald-cutoff                    7.0
~
```

Figure 2: key file for a cubic water box of 49.232 Angstroms using AMOEBA parameters in the water03.prm file

1.5 *.prm files

We saw that a file containing parameters of the force field used is specified using the keyword **parameters** in the key file. This file contains all the parameters of the model, namely global ones such as the type of van der Waals interactions and atom type-specific ones for each potential energy term of the model used: bonded terms and non bonded terms. Furthermore, specific parameters can be added directly in the *key files using the same logic as the one of the parameter file.

```
      ############################
      ##                        ##
      ##  Atom Type Definitions  ##
      ##                        ##
      ############################

atom          1    1    O     "AMOEBA Water O"              8    15.995    2
atom          2    2    H     "AMOEBA Water H"              1     1.008    1

      ################################
      ##                            ##
      ##  Van der Waals Parameters  ##
      ##                            ##
      ################################

vdw           1               3.4050     0.1100
vdw           2               2.6550     0.0135      0.910


      ##################################
      ##                              ##
      ##  Bond Stretching Parameters  ##
      ##                              ##
      ##################################

bond          1    2          556.85     0.9572


      ################################
      ##                            ##
      ##  Angle Bending Parameters  ##
      ##                            ##
      ################################

angle         2    1    2      48.70     108.50


      ############################
      ##                        ##
      ##  Urey-Bradley Parameters  ##
      ##                        ##
      ############################


ureybrad      2    1    2      -7.60     1.5537


      ###################################
      ##                               ##
      ##  Atomic Multipole Parameters  ##
      ##                               ##
      ###################################

multipole     1   -2   -2              -0.51966
                                       0.00000    0.00000    0.14279
                                       0.37928
                                       0.00000   -0.41809
                                       0.00000    0.00000    0.03881

multipole     2    1    2               0.25983
                                      -0.03859    0.00000   -0.05818
                                      -0.03673
                                       0.00000   -0.10739
                                      -0.00203    0.00000    0.14412


      ######################################
      ##                                  ##
      ##  Dipole Polarizability Parameters  ##
      ##                                  ##
      ######################################


polarize      1           0.837      0.390       2
polarize      2           0.496      0.390       1
```

Figure 3: parameter file for the AMOEBA03 water model with definition of the atom types of oxygen and hydrogen and the associated parameters

1.6 Main programs

Tinker 8 is made of various executables, some for production and others for system preparation or analysis/post-processing. Among these, only the ones associated to production have been parallelized and accelerated in the Tinker-HP package. Here, we focus on the Tinker-HP executables and on some of the Tinker 8 ones dealing with system preparation and post processing.

1.6.1 Tinker-HP CPU programs (and outputs)

analyze: Energy decomposition analysis given a xyz file or a trajectory. To be used with the "E" option (even if the **analyze** program of Tinker 8 has more options such as total electrostatic moments calculation or list of the parameters).

Typical command line: **mpirun -np x ./analyze system.xyz e** with **x** the total number of mpi processes you want to run the application with.

```
Total Potential Energy :            -35997.1570 Kcal/mole

Energy Component Breakdown :          Kcal/mole      Interactions

Bond Stretching                       3243.5388           8000
Angle Bending                         1943.9801           4000
Urey-Bradley                           -78.3225           4000
Van der Waals                        18149.9402        4351283
Atomic Multipoles                   -41463.2469         860703
Polarization                        -17793.0467         860703

TINKER-HP is Exiting following Normal Termination
```

Figure 4: typical output of an energy decomposition analysis realized with the **analyze** program with the total potential energy and the various interactions it is the sum of

dynamic: Molecular dynamics using the setup described in the *key file (integrators, thermostat and barostats) and in the command line. More precisely, a typical command line to run a MD with the **dynamic** program is: **mpirun -np x ./dynamic system.xyz nsteps dt dumpfreq ens Temp Press** where:

- **nsteps** is the number of timesteps of the trajectory
- **dt** is the timestep **in femtoseconds** of the dynamics
- **dumpfreq** its frequency of outputs **in picoseconds**
- **ens** is an integer caracterizing the thermodynamic ensemble to sample: 1 for NVE, 2 for NVT, 4 for NPT
- **Temp** is the target temperature in Kelvin
- **Press** is the target pressure in Atmosphere

output of the dynamic program: If the verbose line is in the key file, then the typical output of the execution of the dynamic prorgram will look like:

```
  69   -24942.6347   -35875.8473    10933.8127     305.70    1244.40
  70   -24946.2136   -35900.7588    10954.5452     306.28    1442.61
  71   -24952.7068   -35912.4022    10959.6954     306.42    1117.89
  72   -24959.5105   -35865.3109    10905.8004     304.91     468.93
  73   -24947.8727   -35807.7837    10859.9110     303.63    -232.44
  74   -24940.5802   -35802.5897    10862.0095     303.69    -730.69
  75   -24943.5192   -35862.9412    10919.4220     305.30    -822.14
  76   -24959.3098   -35936.8969    10977.5871     306.92    -447.01
  77   -24958.3118   -35964.5409    11006.2291     307.72     251.30
  78   -24946.4668   -35950.0143    11003.5475     307.65     923.25
  79   -24950.1672   -35949.5669    10999.3997     307.53    1209.35
  80   -24941.6839   -35958.1113    11016.4275     308.01    1015.09
  81   -24937.1816   -35923.8444    10986.6628     307.18     499.77
  82   -24947.0948   -35843.2226    10896.1278     304.64     -89.17
  83   -24955.9278   -35796.4806    10840.5528     303.09    -555.19
  84   -24953.8880   -35835.1763    10881.2884     304.23    -748.08
  85   -24953.0789   -35900.4836    10947.4047     306.08    -563.08
  86   -24962.3579   -35928.1317    10965.7738     306.59     -49.02
  87   -24963.4591   -35915.1552    10951.6961     306.20     558.17
  88   -24976.9682   -35905.6843    10928.7161     305.56     931.73
  89   -24990.1324   -35930.4470    10940.3147     305.88     883.11
  90   -24981.1072   -35941.7593    10960.6521     306.45     497.51
  91   -24976.7192   -35869.1696    10892.4504     304.54      12.68
  92   -24972.6712   -35782.6427    10809.9715     302.24    -387.96
  93   -24981.2167   -35790.9214    10809.7047     302.23    -608.59
  94   -24987.3084   -35871.2198    10883.9114     304.30    -540.65
  95   -24993.4268   -35943.7333    10950.3065     306.16    -170.10
  96   -24988.7148   -35961.9725    10973.2577     306.80     352.40
  97   -24994.8547   -35938.5537    10943.6990     305.97     751.12
  98   -24992.6670   -35940.7596    10948.0927     306.10     803.56
  99   -24998.3453   -35977.7128    10979.3674     306.97     514.33
 100   -25011.6175   -35943.8932    10932.2757     305.66     125.91

 Average Values for the last   100 out of      100 Dynamics Steps

 Simulation Time              0.1000 Picosecond
 Total Energy            -24919.7286 Kcal/mole   (+/-  45.5456)
 Potential Energy        -35864.5745 Kcal/mole   (+/-  91.3139)
 Kinetic Energy           10944.8459 Kcal/mole   (+/-  90.6691)
 Temperature                 306.01 Kelvin      (+/-     2.54)
 Pressure                    136.64 Atmosphere  (+/-  1077.85)
 Density                     1.0024 Grams/cc    (+/-   0.0001)
 ==================================
Ave time for pos./vel. update     =        0.001147 seconds          0.079%
Ave time for reneigh              =        0.001050 seconds          0.072%
Ave time for pos. comms.          =        0.000907 seconds          0.062%
Ave time for param regeneration   =        0.001558 seconds          0.107%
Ave time in gradient routines     =        1.386223 seconds         95.042%
Ave time for neighbor lists       =        0.053337 seconds          3.657%
Ave time for energy reduction     =        0.008797 seconds          0.603%
Ave time for Temp/Press control   =        0.002991 seconds          0.205%
Ave time for forces comms.        =        0.001793 seconds          0.123%
Ave time per step                 =        1.458532 seconds        100.000%
 ns per day:          0.0592
 ==================================

 TINKER-HP is Exiting following Normal Termination
```

Figure 5: typical output of an md run realized with the **dynamic** program. First, each line contains informations about the state of the system at the timesteps: timestep, total energy, potential energy, kinetic energy, instantaneous temperature and pressure. Then some statistic about the last 100 timesteps is printed: average energies, temperature, pressure... as well as some timings

Additionnaly, at the frequency chosen in the command line, the current geometry of the system is printed out in a separate file. If nothing is specified, each geometry (with the tinker xyz format) will be printed in a separate file (**system.001, system.002...**), but by putting the keyword **archive** in

the keyfile, these will be concatenated in a single **system.arc** file. Moreover, if the keyword **dcdio** is put in the keyfile, then these will be written with the CHARMM dcd (binary and so much lighter) format in the file **system.dcd**. Note that such files can be read with software handling this format such as VMD by using the xyz file as a topology file. All these can be further processed by the **analyze** program.

Furthermore, a restart file **system.dyn** is written in the same directory. **Note that if there is such a file in the same directory then the trajectory will start from the associated state.**

minimize: Geometry optimization with a L-BFGS quasi Newton algorithm. A typical command line is **mpirun -np x ./minimize system.xyz cvcrit** with cvcrit the convergence criterion of the minimization procedure (the smaller the tigther convergence), typically 0.1 is usually enough to remove steric clashes before running a MD. The program will write the associated optimized geometry in a file **system.xyz**_2.

```
Nx =     2  Ny =     2  Nz =     2
   828   -49379.3036     0.1317     0.2499    0.0012   80.24    913    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   829   -49379.6142     0.1671     0.3105    0.0010   72.79    914    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   830   -49379.9393     0.2076     0.3252    0.0016   78.50    915    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   831   -49380.2289     0.1298     0.2896    0.0010   79.47    916    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   832   -49380.4458     0.1200     0.2169    0.0005   65.46    917    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   833   -49380.6631     0.1556     0.2173    0.0010   76.37    918    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   834   -49380.9495     0.1499     0.2864    0.0010   76.25    919    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   835   -49381.2346     0.1846     0.2851    0.0020   81.20    920    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   836   -49381.4840     0.1497     0.2494    0.0007   73.03    921    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   837   -49381.6735     0.1208     0.1896    0.0008   78.99    922    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   838   -49381.8876     0.1309     0.2141    0.0006   64.17    923    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   839   -49382.0857     0.1255     0.1981    0.0010   78.35    924    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   840   -49382.2912     0.1181     0.2055    0.0006   72.82    925    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   841   -49382.5077     0.1528     0.2165    0.0011   74.09    926    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   842   -49382.6835     0.1118     0.1758    0.0006   74.04    927    Success
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2
   843   -49382.8056     0.0942     0.1220    0.0004   69.64    928   SmallGrad

 LBFGS  --  Normal Termination due to SmallGrad
 3D Domain Decomposition
Nx =     2  Ny =     2  Nz =     2

 Final Function Value :       -49382.8056
 Final RMS Gradient :              0.0942
 Final Gradient Norm :            10.3205

 TINKER-HP is Exiting following Normal Termination
```

Figure 6: typical output of a geometry optimization realized with the **minimize** program

testgrad: Tinker-HP also comes with a parallel version of the **testgrad** program that gives analytical and/or numerical derivatives of the separate energetic contributions to the total potential energy of a system. This is useful for debugging purposes or in a situation where one needs to get these forces for example in a QM/MM simulation where these are fed to a QM program.

1.6.2 Tinker-HP GPU programs

1.7 additional Tinker 8 (CPU) programs)

The tinker suite comes with several pre or post-processing executables, even if the output formats (***.arc** and ***dcd**) are compatible with several post-processing tools such as tha **ambertools** or **mdtraj**.

Among these, **xyzedit** is especially useful as it allows (among many possibilities) to solvate a molecule in a box of solvent, to add counterions to a charged system to neutralize it or to change an atom type into another.

```
     ######################################################################
   ##########################################################################
  ###                                                                    ###
 ###            Tinker  ---  Software Tools for Molecular Design           ###
 ##                                                                         ##
 ##                      Version 8.10.2   February 2022                    ##
 ##                                                                         ##
 ##               Copyright (c)  Jay William Ponder  1990-2022              ##
 ###                           All Rights Reserved                        ###
  ###                                                                    ###
   ##########################################################################
     ######################################################################


 The Tinker XYZ File Editing Utility Can :

    (1) Offset the Numbers of the Current Atoms
    (2) Remove User Specified Individual Atoms
    (3) Remove User Specified Types of Atoms
    (4) Delete Inactive Atoms Beyond Cutoff Range
    (5) Insertion of Individual Specified Atoms
    (6) Replace Old Atom Type with a New Type
    (7) Assign Connectivities for Linear Chain
    (8) Assign Connectivities Based on Distance
    (9) Assign Atom Types for BASIC Force Field
   (10) Convert Units from Bohrs to Angstroms
   (11) Invert thru Origin to Give Mirror Image
   (12) Translate All Atoms by an X,Y,Z-Vector
   (13) Translate Center of Mass to the Origin
   (14) Translate a Specified Atom to the Origin
   (15) Translate and Rotate to Inertial Frame
   (16) Rotate All Atoms Around a Specified Axis
   (17) Move to Specified Rigid Body Coordinates
   (18) Move Stray Molecules into Periodic Box
   (19) Trim a Periodic Box to a Smaller Size
   (20) Make Truncated Octahedron from Cubic Box
   (21) Make Rhombic Dodecahedron from Cubic Box
   (22) Append a Second XYZ File to Current One
   (23) Create and Fill a Periodic Boundary Box
   (24) Soak Current Molecule in Box of Solvent
   (25) Place Monoatomic Ions around a Solute

 Number of the Desired Choice [<Enter>=Exit] :
```

Figure 7: options of the **xyzedit** program

Also, the **radial** program can be used to compute radial distribution functions that can be then compared to experimental data. A typical corresponding command line is:

./radial system.arc first-frame last-frame frame-incr type1 type2 dr Y/N

where **first-frame** and **last-frame** are the indexes of the first and the last frame of the trajectory to be considered, **frame-incr** the frame increment (typically 1), **type1** and **type2** the 2 atom types to be considered, **dr** the distance discretization in Angstroms (typically 0.01), Y/N a flag governing the account for intramolecular distances or not.

2 Installation of Tinker-HP

2.1 Installation for CPUs

2.1.1 Prerequisites

Tinker-HP requires the MKL library, a FFT library (such as FFTW) and a slightly modified 2DECOMP_FFT library (shipped with Tinker-HP) in order to run. The 2DECOMP_FFT library enables parallel 3D FFT computations based on 2d-pencils data distribution (see 2DECOMP_FFT site) based on a sequential implementation of FFTs such as the one provided by the FFTW library.

Tinker-HP also requires a recent enough MPI library supporting MPI 3.x standards such as non blocking collectives. The code has been extensively tested with recent IntelMPI versions (such as intel MPI 5.1) and better performances have been observed with this family of MPI implementation compared to other ones such as OpenMPI .

As Tinker-HP is shipped in source form, you need to compile it. This was not always an easy task in the previous releases. Tinker-HP now uses a `configure` script built with autotools packages from GNU to ease the compilation and installation process. The first thing you should do is to type :

`autoconf ; automake`

in the main directory. This will generate the `configure` file. Apart from the usual options available with all `configure` scripts, there are specific options for Tinker-HP.

```
Usage: ./configure [OPTION]... [VAR=VALUE]...

Optional Features:
  --enable-debug                Enable debug mode (check array bounds, implicit
                                none, etc...). Should not be active in normal
                                operations [default is no]
  --enable-skylake              Enable AVX512 Optimization for Skylake Processors
                                [default is no]
  --enable-knl                  Enable AVX512 Optimization for KNL (Xeon Phi)
                                Processors [default is no]
  --enable-fft-generic          Enable generic FFT mode [default is yes]
  --enable-fft-mkl              Enable MKL   FFT mode [default is no]
  --enable-fft-fftw3            Enable fftw3 FFT mode [default is no]
  --enable-fft-fftw3_f03        Enable fftw3_f03 FFT mode [default is no]
  --enable-plumed               Enable plumed interface [default is no]
  --enable-colvars              Enable Colvars interface [default is no]
Optional Packages:
  --with-blaslib=<BLAS LIB>     Specify BLAS library [mkl, lapack or
                                /absolute/path/to/BLAS_library]
  --with-fftlib=<FFT LIB>       Specify a library for FFT called by 2decomp [mkl or
                                fftw3 or /absolute/path/to/FFTW_library]
```

The ultimate goal of this script is to let you type

```
./configure ; make ; make install ; cd example ; ./ubiquitin2.run
```

and have everything compiled, installed and running.

2.1.2 List of Options

As for all the `configure` scripts, you can choose the directory in which the binaries will be copied. So, `configure` has `--prefix=<DIR>`.

Tinker-HP has a special interest to know if it will run on AVX-512 capable processors. So, `configure` has options for that:

```
--enable-slylake

--enable-knl
```

Recall that Tinker-HP needs to make a FFT decomposition with a modified version of the 2DECOMP_FFT library, which in turn needs a working FFTW library. This is why you can find `configure` options about FFT interface and library:

```
--enable-fft-generic

--enable-fft-mkl

--enable-fft-fftw3

--enable-fft-fftw3_f03

--with-fftlib=<FFT LIB>
```

Tinker-HP also needs some functions that resides in a working BLAS library. So, there is an option for that:

```
--with-blaslib=<BLAS LIB>
```

Tinker-HP is now able to be interfaced with PLUMED. So, there is an option for that :

```
--enable-plumed
```

Tinker-HP is now able to be interfaced with COLVARS. So, there is an option for that :

```
--enable-colvars
```

Finally, as there might be some execution problems, or compilation problems for the users who develop code, Tinker-HP has an `--enable-debug` option.

`configure` tries to find its path to reach a valid MPI compiler and a valid Fortran compiler by unsetting the environment variables `$FC` and `$F77`, and reading the environment variable `$PATH`. It also tries to find valid FFT and BLAS libraries by reading `$FFTW` and `$MKLROOT` or `$LAPACK` respectively. Most of the time, these environment variables are defined through the `module` framework. As a try, do a

```
module available 2>&1 | less
```

to see if you have the `module` framework installed on your machine, and to know what `module` you can load. `configure` then figures out how to build the correct Makefiles.

By default, `configure` chooses the MKL library from Intel as the BLAS and FFTW3 libraries, sets the `--enable-fft-mkl` option, does not make any processor optimization, and disables debugging. Thus, typing `./configure` give the same result as if you have typed `./configure --enable-fft-mkl --with-blaslib=mkl --with-fftlib=mkl`.

2.1.3 Using `configure`

If you want to have different settings than those used by default, you'll have to give `configure` more information. Be aware that `configure` cannot magically guess anything. So, the information you give must be precise and complete.

Install Directory By default, this is where you have unzipped and untarred the distribution. If you want another place, use `--prefix=<DIR>`. You can choose any directory you want, providing that you have permission to create this directory and/or write in it.

Processor optimization The machine on which you compile is not always the one on which Tinker-HP will run. If you know that Tinker-HP is going to run on AVX-512 capable processors, you are strongly encouraged to use one of :

`--enable-knl` for KNL processors (also known as Xeon-Phi)

`--enable-slylake` for Skylake processors.

as this will dramatically improve the execution speed. Otherwise, the optimization will be done using the capabilities of the compilation machine, as determined by the compiler.

FFT interface You can choose the interface of FFTW you want to use. This has an effect on the 2DECOMP_FFT library. So :

`--enable-fft-generic` gives the generic FFT, with no call to FFTW library

`--enable-fft-mkl` gives the MKL FFT. It also automatically selects the MKL library as the FFTW library.

`--enable-fft-fftw3` gives the fftw3 interface, and is designed to work with an external FFTW library.

`--enable-fft-fftw3_f03` gives the fftw3 Fortran2003 interface, and is designed to work with an external FFTW library

FFT Library You can choose the FFTW library you want to use. It can come from the MKL suite, or some FFTW3 package (either system installed, or compiled by you). So :

`--with-fftlib=mkl` : selects the MKL library, but needs the variable `$MKLROOT` to be set to the absolute path of the MKL library

`--with-fftlib=/path/to/mkl/library` : selects the MKL library by giving the absolute path of the MKL library

`--with-fftlib=fftw3` : selects the FFTW3 library, but needs the variable `$FFTW` to be set to the absolute path of the FFTW3 library

`--with-fftlib=/path/to/fftw3/library` : selects the FFTW3 library by giving the absolute path of the FFTW3 library

Here are typical commands you can type. If `$MKLROOT` has been correctly set :

```
./configure --enable-fft-mkl --with-fftlib=mkl
```

If you wish to give the absolute path of the library :

```
./configure --enable-fft-mkl   --with-fftlib=/path/to/mkl/library
./configure --enable-fft-fftw3 --with-fftlib=/path/to/fftw3/library
```

These last commands can also be written this way :

```
MKLROOT=/path/to/mkl/library ./configure --enable-fft-mkl   --with-fftlib=mkl
FFTW=/path/to/fftw3/library  ./configure --enable-fft-fftw3 --with-fftlib=fftw3
```

BLAS library You can choose the BLAS library you want to use. It can come from the MKL suite or some LAPACK package (either system installed, or compiled by you). So :

`--with-blaslib=mkl` : selects the MKL library, but needs the variable `$MKLROOT` to be set to the absolute path of the MKL library

`--with-blaslib=/path/to/mkl/library` : selects the MKL library, by giving the absolute path of the MKL library

`--with-blaslib=lapack` : selects the LAPACK library, but needs the variable `$LAPACK`to be set to the absolute path of the LAPACK library

`--with-blaslib=/path/to/lapack/library`: selects the LAPACK library, by giving the absolute path of the LAPACK library.

Here are typical commands you can type. If `$MKLROOT` has been correctly set :

```
./configure --enable-fft-mkl --with-blas=mkl
```

If you wish to give the absolute path of the library :

```
./configure --enable-fft-mkl --with-blas=/path/to/mkl/library
./configure --enable-fft-mkl --with-blas=/path/to/lapack/library
```

These last commands can also be written this way :

```
MKLROOT=/path/to/mkl/library    ./configure --enable-fft-mkl --with-blas=mkl
LAPACK=/path/to/lapack/library  ./configure --enable-fft-mkl --with-blas=lapack
```

2.2 Installation for GPUs

2.2.1 Prerequisites

Hardware A relatively recent Nvidia GPU is mandatory for the GPU code. One with at least a compute capability above 5.0.

Compilers and OS

- Linux is the recommended Operating System to run Tinker-HP. The Windows subsystem for Linux (WSL) may run Tinker-HP but it is known to be unstable due to compatibility issues between the software and the GPU pilot.
- A single package gathers every tools required to compile and execute Tinker-HP. The Nvidia HPC Package is the package for the job. Any version under 21.2 ! is operational. An issue involving "atomic operations" has been discovered with higher versions of the package (21.[3-9]). Luckily it is possible to bypass it by adding a compiler flag during the build step. Look for `add_options_f` variable configuration inside ci/install.sh to be revert to the old behavior and fix the problem. However be aware that this is just a temporary fix. Starting from version `22.XX`, the previously described issue has been solved by the developers, and the OpenACC compiler bug lately detected within 22.XX version has also been fixed. It involves an ambiguous OpenACC compilation when calling an acc device subroutine compiled in an other translation unit. This package contains everything you need to build Tinker-HP (GPU code). However, we need to make sure the installation has been correctly done and the environment variables (PATH; CPATH & NVHPC) are set and updated

Python Environment If you plan to combine machine learning with MD, we provide an environment file `tinker-hp/GPU/tinkerml.yaml` which installs python modules for machine learning. This specific python environment requires Anaconda or Miniconda to be installed on the host machine. Install the environment according to the next instructions before proceeding to Tinker-HP.

To create the environment, run in your shell: `conda env create -f tinkerml.yaml;` Activate or deactivate it respectively with `conda activate tinkerml` or `conda deactivate;`

Composition of the environment:

- Pytorch, TensorFlow are the building block of most of machine learning potential libraries.
- Deepmd-kit and libdeepmd are used for DeePMD models.
- Our TorchANI-based library composed of lot of new features, more coming soon!

Mandatory Libraries

- FFt libraries
  `libfftw3` or `libfftw3f` in single or mixed precision mode. You will need to provide in that case `FFTW` install directory to both Tinker's and 2decompfft's Makefile. Setting `FFTW=/path/to/fftw3` as an environment variable should be enough to make. It is also possible to use the generic fft provided by the 2decomp_fft source code (Default behavior). It is better to sitck to the default behaviour, since this concern an unused portion of the application.
- Intel MKL Library (Only for host/CPU build!)
  Set `MKLROOT=/path/to/mkl/library` before make
- 2decomp_fft library is already provided inside the repository.

- Standard C++ library
  Set `GNUROOT=/path/to/gnu`. This is not mandatory for a first and unique installation with `ci/install.sh` but it is required for developers or when you need to recompile in an other configuration.
  e.g. `export GNUROOT=/usr` for conventional Linux systems. The aim is to fetch `libstdc++` from `$GNUROOT/lib64`

- CUDA Libraries. *cuFFT, cuBlas, cuSolver, cuRand* are used to generate executable and are already present within the NVIDIA HPC-SDK package.

2.2.2 Quick Build

For most users, we recommend this relatively accessible method which lies on a bash installation script named `/tinker-hp/GPU/ci/install/sh`. After setting your environment according to the Prerequisites, change directory to the GPU folder, edit the configuration section of the installation script to select your build and proceed with the commands below.

```
$> pwd
#> $HOME/.../tinker-hp/GPU
$> ./ci/install.sh
```

List of options The configuration section of `ci/install.sh` lists some variables which are passed to the Makefile during the compiling stage. Edit this part according to your need, before executing the script. We keep this section updated as the development progresses.

`dbuild`	: for out-of-source build. If interested in an other configuration without losing the previous object files, you may use a different value like `$tinkerdir/build0`
`target_arch`	: control the target platform architecture. Decide whether the build should target CPUs or GPUs
`c_c`	: Stand for compute capability. It is specific to the GPU device. Default option target every Nvidia GPU from Pascal architecture to Ampere.
`build_plumed`	: (switch) enables/disables PLUMED interface. PLUMED is automatically built
`build_colvars`	: (switch) enables/disables COLVARS interface. COLVAR is automatically built
`NN`	: (switch) enables/disables Neural Network python interface.
`FPA`	: (switch) enables/disables fixed precision arithmetic in mixed precision (enabled by default) Requires a CUDA python environment consistent with the installation one before building.

This script will create and install two types of binaries for every applications. Double and mixed precision program will be available by the end of the installation, into the installation directory. As long as you stand for one configuration at the time, The installation script will serve as an ideal build assistant. In the event of the installation script failure, we recommend to follow one the procedures given below.

2.2.3 Build with Makefile

This part concerns every developers or anyone familiar enough with source code compilation. `GPU/source/Makefile.pgi` allows every type of build. By editing configurations variables you can change default compilers, adapts flags, rename libraries, and so on. Here is the description of the useful variables need to configure your own build's.

configurations variables

`NVTX_SUPPORT`	enables NVTX markers during profiling if set to 1. This option is not useful in release construct.
`NVSHMEM_SUPPORT`	enables build with nvshmem library if set to 1. This option requires NVSHMEM_HOME variable to be set on nvshmem install directory. It has not been tested with recent Nvidia HPC package.
`NO_MUTATION`	disables soft-core computation if set to 1
`FPA_SUPPORT`	enables fixed point arithmetic in non-Bonded Forces and energy reduction if set to 1. It requires mixed precision to be enabled through the Makefile precision variable.
`PLUMED_SUPPORT`	builds and link both Tinker-HP and Plumed Package supplied next to it.
`NN_SUPPORT`	builds, interfaces and links Tinker-HP with machine learning code
`arch=(host|[device])` `.or. (cpu|[gpu])`	allows you to select target architecture
`prec=(single|mixed|[double])` `.or. (s|m|[d])`	changes the precision build
`prefix=(path/|[../bin])`	controls the installation directory
`prog_suffix=(any_string|[])`	append suffix to install binary. This variable may allow you to have multiple binaries in the installation directory. For instance analyze or analyze.gpu `compute_capability=(list|[60,70])` selects the device's compute capability for the construct. this variable shall accept a list of compute capability separated by a comma. e.g. `compute_capability=35,60,75,80`
`cuda_version=([11.0])`	contains the cuda version to be used for construct. PGI 19.10 compilers only support cuda 9.2 10.0 and 10.1 for OpenACC. Latest NVIDIA HPC-SDK comes with either a unique cuda version or multiple versions from 11.0 to 12.0. To be consistent with the construct, we strongly recommend to use the same version of CUDA compiler as your OpenACC compiler.
`opt=(debug|debug+|[release])`	decides the optimization level of the compilation

Custom Build Let suggest that we want CPU binaries in double precision combined with an out-of-source build by hand. After linking source/Makefile.pgi to source/Makefile, we should follow the next script for an out-of-source build :

```
$> pwd
#> /home/$user/.../tinker-hp/GPU
$> mkdir -p bin
$> cd source
$> make arch=cpu 2decomp_fft_rebuild        # build lib2decompfft.a
$> make thrust_lib_rebuild                  # build libwapper.a
$> make create_build BUILD_DIR=../build_CPU # create build directory
$> cd ../build_CPU
$> make arch=cpu all -j6              # build Tinker's program for CPU
$> ls ../bin                                # list binaries
#> analyze dynamic minimize bar pimd
```

You can also create a configuration bash file that store your configuration and run it with your desire targets.

```
$> cd source
$> cat << EOF > C.sh
#!/bin/bash
make FPA_SUPPORT=1 COLVARS_SUPPORT=1 prec=mixed arch=device\
     compute_capability=60,70 cuda_version=11.3\
     prog_suffix=.gmix $@
EOF
$> . C.sh 2decomp_fft_rebuild  # Rebuild 2decomp_fft
                               # library in single precision
$> . C.sh -j8                  # Build binaries (*.gfix)
                               # for 60,70 compute_capability
                               # device using fixed precision
```

2.2.4 Run through Docker

In addition to GitHub, a GPU container (quick install!) is available thanks to NVIDIA on the NVIDIA NGC's website.

3 First examples: liquid water simulations

3.1 AMOEBA simulations

- start from **waterhuge.xyz**
- vizualize the system with **VMD**
- verify the **waterhuge.key** file
- run an energy decomposition analysis on this system
- run a minimization of energy on this system
- run a short (100 picoseconds) NPT simulation at 300K and visualize it with VMD
- run a 500ps long NVT simulation
- compute the Oxygen-Oxygen, Oxygen-Hydrogen and Hydrogen-Hydrogen radial distribution functions of AMOEBA water using **radial**

3.2 SPC simulations

- convert the waterhuge.xyz file to a water box of the SPC water model using **xyzedit**
- re-run a minimization and a 100ps NPT simulation
- get the same radial distribution functions for this other water model and compare them

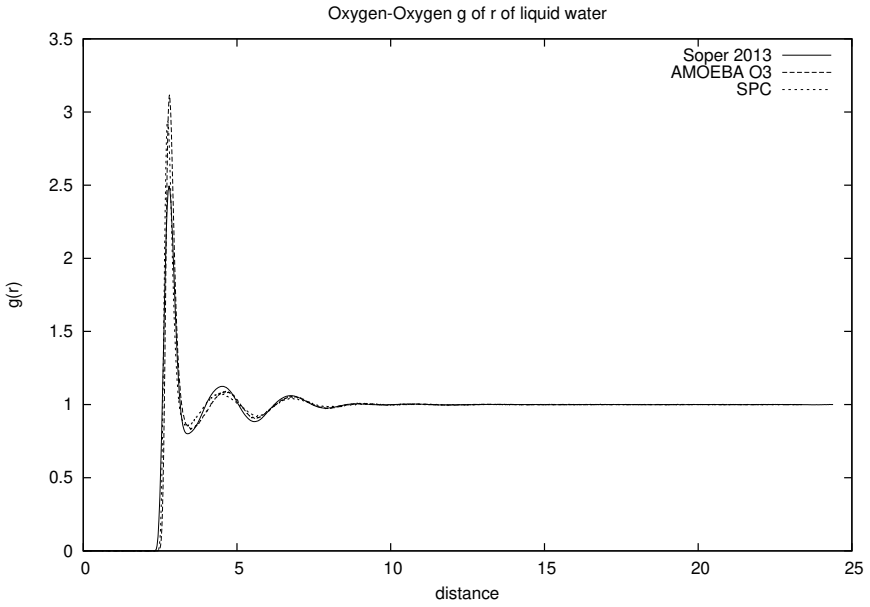

Figure 8: Oxygen-Oxygen radial distribution function

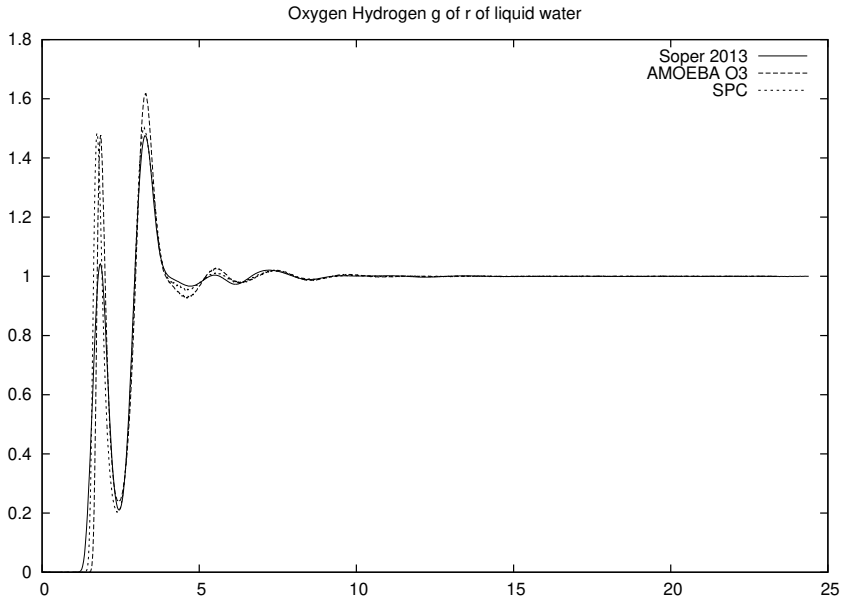

Figure 9: Oxygen-Hydrogen radial distribution function

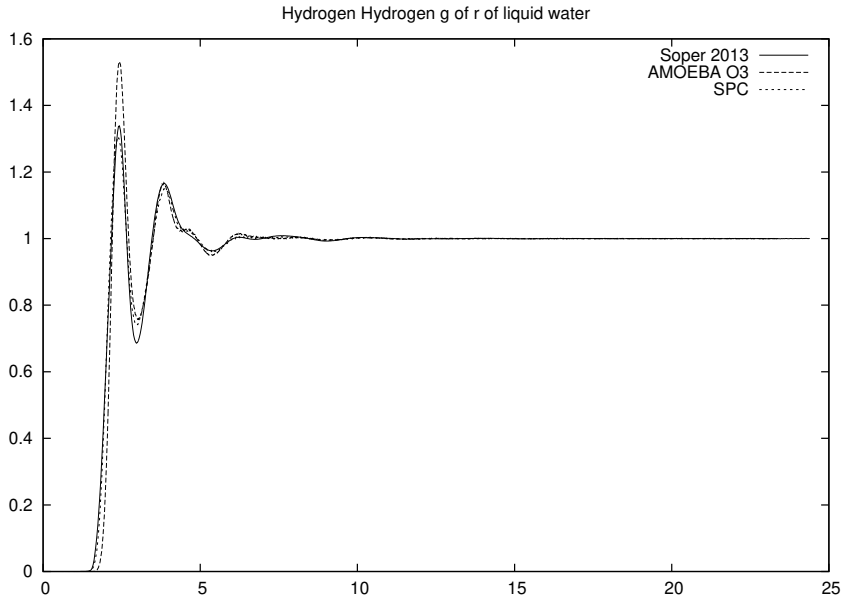

Figure 10: Hydrogen-Hydrogen radial distribution function

4 Second example: solvated protein, from system preparation to MD in the NPT ensemble

- get the 1FSV protein motif on the pdb website
- convert it to the Tinker xyz format using **pdbxyz**
- convert the atom types of **waterhuge.xyz** to be compatible with **amoebabio18.prm**
- solvate **1fsv.xyz** using **waterhuge.xyz_2** and **xyzedit**
- run a constant pressure/constant temperature simulation of 500 picoseconds at 1 atmostphere
- visualize the trajectory with VMD
- compute the associated RMSD using VMD

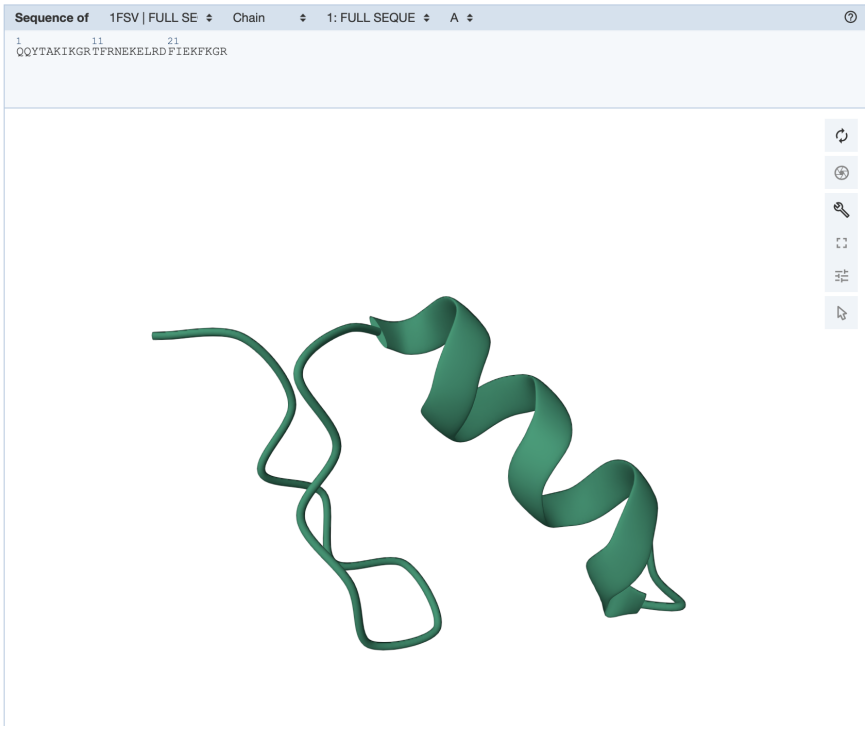

Figure 11: 1FSV protein as can be seen on the pdb website

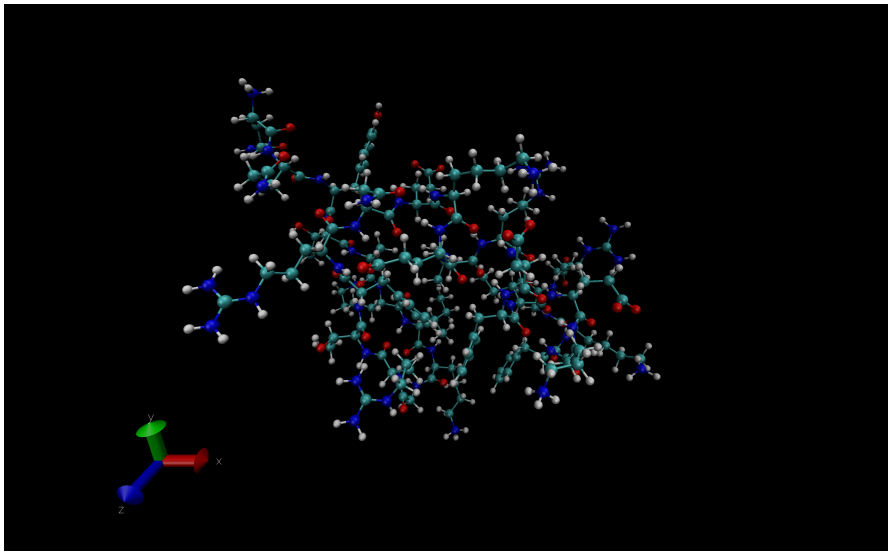

Figure 12: 1FSV protein visualized in VMD with the CPK representation

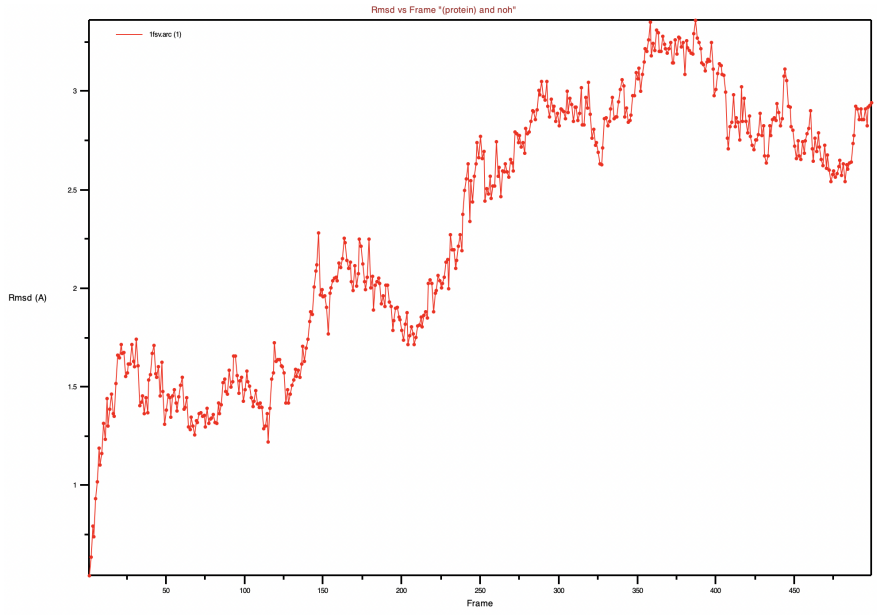

Figure 13: RMSD of the 1FSV protein during a 500ps MD trajectory computed with VMD

5 Third example: enhance sampling simulation, PHI/PSI free energy alanine dipeptide free energy landscape through ABF simulations using colvars

5.1 Enhanced sampling and collective variables

Probability distributions corresponding to the potential energy surface of molecular systems in a canonical ensemble are typically associated with metastable states separated by barriers such that a simple MD trajectory would never visit these basins in a reasonable computational time. There are a lot of methods designed to accelerate the sampling of such distribution in a supervised or unsupervised way. A recent and complete review of these techniques can be found here.

If there is a function $\xi(q)$ (called **reaction coordinate** or **collective variable**) of the cartesian coordinates, typically of much smaller dimension of the complete degrees of freedom $3N$ that "explains" the change between the metastable states, then specific methods to enhance the sampling along this variable can be applied and the metastable can be visited in a reasonable time. These reaction coordinates can be structure based (a distance, an angle, a dihedral...) or more complex such as in alchemical free energy computations.

These methods usually come with a **free energy estimator** to compute the free energy associated with the reaction coordinate, which is also called the **potential of mean force** (**PMF**) that give the free energy barrier associated to the basin changes.

5.2 Adaptive Biaising Force

Several packages, such as **Colvars** or **PLUMED** can be compiled and linked with molecular dynamics package in order to use such accelerated sampling techniques. Here we will be using **Colvars** with the **Adaptive Biasing Force** (**ABF**) method.

To use **Colvars** with Tinker-HP, one needs to add a colvars input in the working directory with a ***.colvars** extension.

See here for a complete description of the capabilities of **Colvars** in conjunction with **NAMD** that are similar to the ones accessible with **Tinker-HP**.

Without going into too many details that would be out of the scope of this school, **ABF** adaptively learns the mean force along the reaction coordinate (discretized in bins) and biases the dynamics with this evolving estimate so that the associated free energy landscape is more and more flattened.

5.3 Application to the Φ-Ψ angles of the alanine dipeptide

A well known (and simple) example of a situation where sampling along a set of reaction coordinates need to be accelerated in order to visit metastable states is the alanine dipeptide. More precisely, the Φ and Ψ Ramachandran are coupled in such a way that several metastable basins separated by high barriers exist. The object of this section of the Winter-School is to use Tinker-HP in conjunction with Colvars to get the 2 dimensional free energy profile of these two angles.

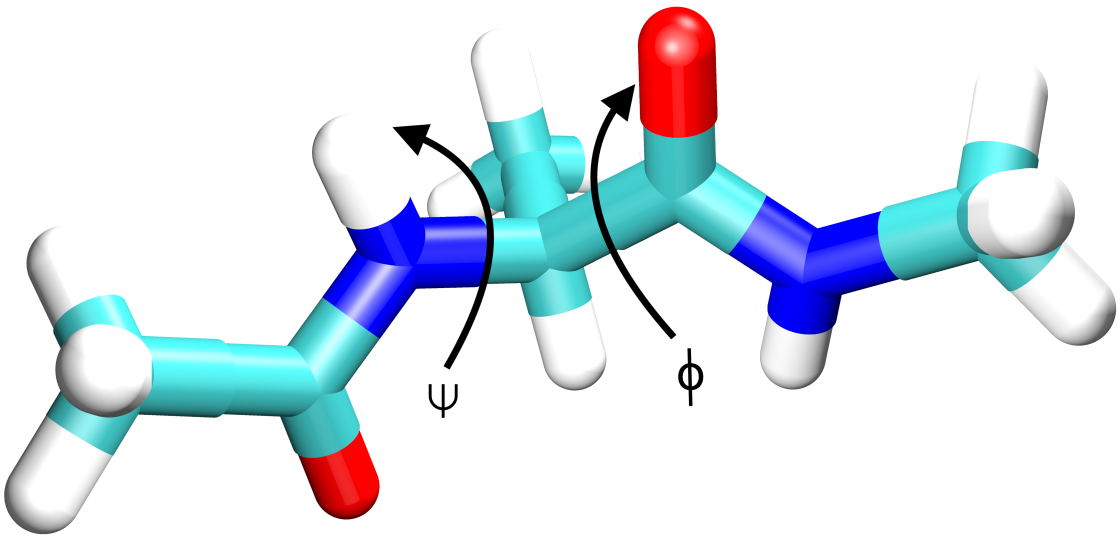

Figure 14: representation of the Ramachandran angles of the alanine dipeptide

- visualize the system **aladim_amber.xyz** with VMD
- get the atoms associated to the two angles and verify the colvars input **aladim_amber.colvars**
- run a 1ns ABF trajectory of this system
- while it runs, visualize the histogram of the Φ Ψ angles in the file **aladim_amber.count** and the associated PMF in the file **aladim_amber.pmf**

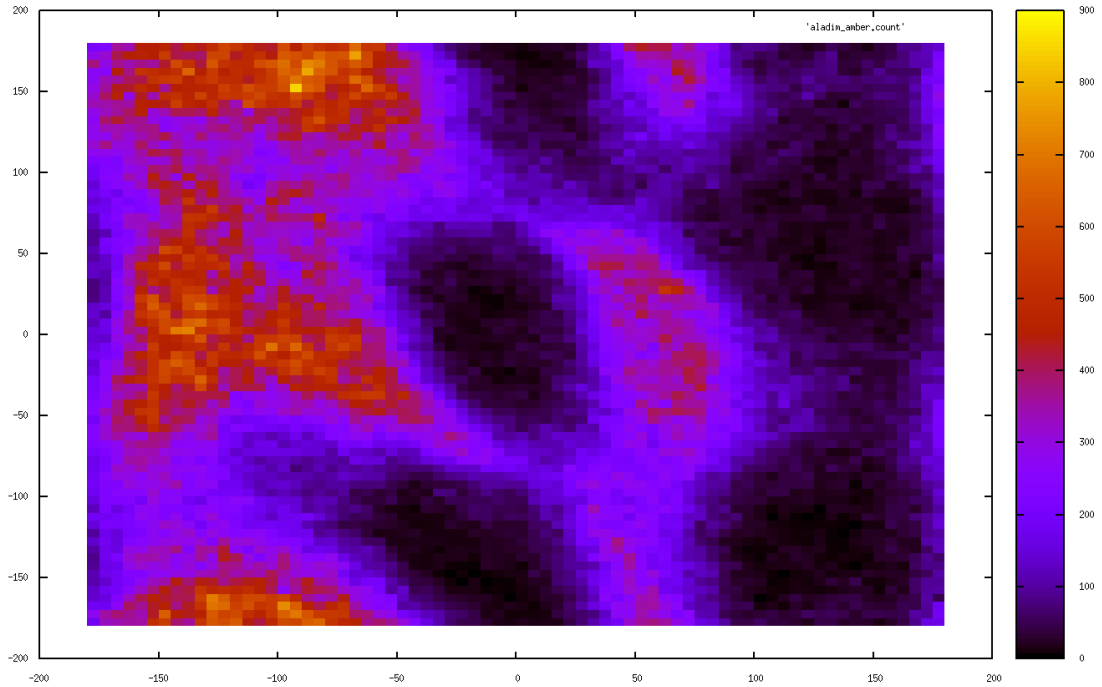

Figure 15: 2D histogram of an ABF simulation to sample the Φ Ψ Ramachandran angles of the alanine dipeptide with the amber force field

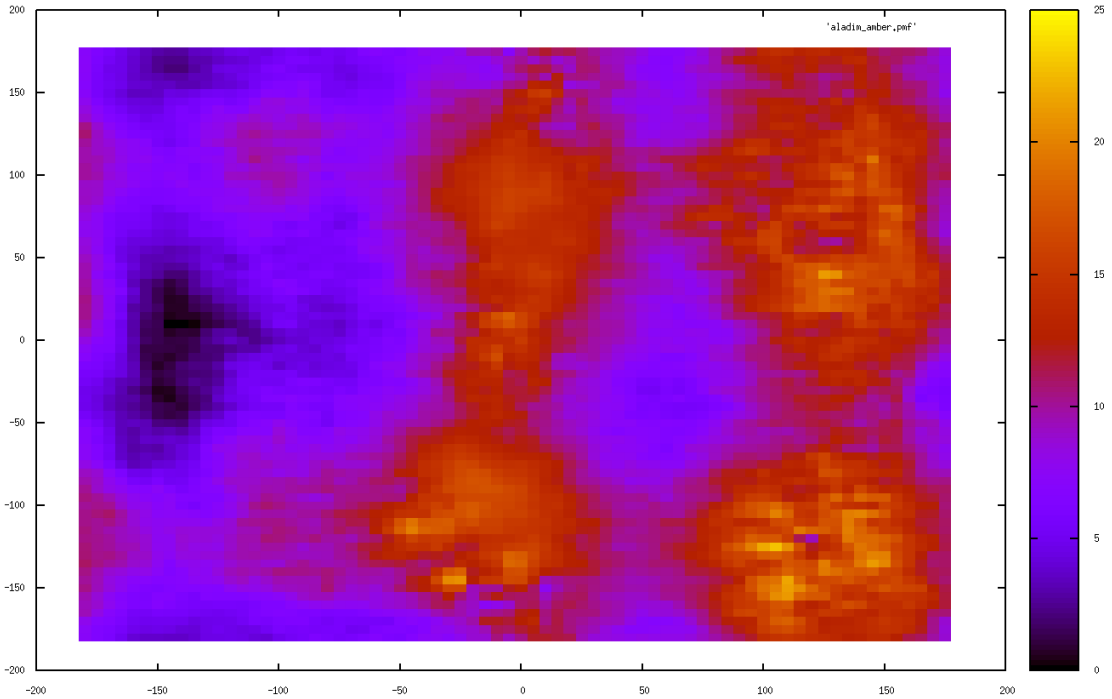

Figure 16: 2D Φ Ψ (Ramachandran angles) PMF (Potential of Mean Force) from an ABF simulation with the amber force field

References