TrajXCUDA - A GPU based trajectory analysing tool

Suresh Kondati Natarajan

December 4, 2015

1 INTRODUCTION

General purpose graphical processing units (GPGPUs) are recently used for accelerating data parallel regions of a code more efficiently. TrajXCUDA is a CUDA based toolkit for processing trajectories from molecular dynamics simulations. However, desktop GPUs often have small on board memory in the range 2 - 4 Gbs, which is not enough to process huge trajectories. In this case, the trajectories have to be split into parts, processed individually and then averaged. Lets hope that with next CUDA release we get over booking memory options, which will make this split and stitch operation obsolete. The rest assured, for the same data GPUs outperform CPUs by a factor of 10 or even more based on the number of cores and clock frequency of the GPU.

2 INSTALLATION

Download the tar ball and extract it in a folder. This program can be compiled with Nvidia's nvcc compiler and needs CUDA runtime and CUDA toolkit installed. Make sure you have fftw3 installed as well. There is a Makefile in the folder, set the environmental variables and paths before typing make to install the package. If you wish to reinstall, please type make clean before typing make again.

3 KEYWORDS

3.1 Mandatory and general keywords

scope option

This keyword defines how the trajectory will be processed. The trajectory can be stored as a whole by using the option `bulk` or just the atoms that were present at the start of the simulation at the interface or bulk region by using the option `bulk-interface`. If `bulk-interface` option is set, few other keywords must also be set, namely, **max_bond_distance** to specify the bulk-interface separation criterion and **plot** to choose the interfacial atoms or bulk atoms.

compute option

This keyword specifies the type of calculation to be done. The options are as follows.

`density`	to get density plot - side view.
`exchange`	to get the number of atoms exchanged between interface and bulk
`density-top`	to get density plot - top view.
`densityvelocity`	to get density and velocity plots - side view.
`rdf`	radial pair distribution functions
`O-fragments`	counting the fragments of O species.
`percentage-coverage`	to get percentage surface coverage of a species.
`diffusion-coefficient`	to get diffusion coefficient from MSD.
`VDOS`	to obtain VDOS spectrum.
`Z-dist`	to get average density of a atom species along surface normal.
`Hbonds`	to compute number of hydrogen bonds, its angles and distances.

element_1 option

This keyword specifies the atom species used for the analysis. The options are the element symbol or simply `all`. The option `all` is used currently for reading velocities only.

element_2 option

For some **compute** options, this keyword must be set to another element symbol. Used for **compute** options `RDF` and `Hbonds`.

skip_frames option

This allows to skip the `option` number of frames for every frame stored from the trajectory. Be careful for `VDOS` **compute** option.

cell_type option

Two cell types are allowed in the package. The options are `orthorhombic` and `monoclinic`. These option needs in a single line the lattice information on the last line of the trajectory file. For `orthorhombic` it is `xx yy zz` and for `monoclinic` it is `xx yy zz yx`.

metal_species option

The surface metal species must be entered for local distance searches to split interfacial and bulk species. The element symbol is used for the option.

plot option

The default is `all`. However, with **scope** set as `bulk-interface`, you can choose `int` for interfacial atom and `bulk` for bulk atoms. This is currently working for side view density plots.

4 TOOLS

In this section, **compute** specific keywords will be listed and additional information for different **compute** schemes will be provided. For all the **compute** schemes the input files are the same, namely, `input.xyz` with the lattice info in the last line, which has the Cartesian position information, `inputvel.xyz`, which has the Cartesian velocity information. Right now the velocity file is in the LAMMPS output format and the position file is in simple XYZ format. The control file is named `input.dat`, which must contain all the mandatory and necessary keywords for each **compute** scheme.

4.1 compute density

This tool will split the entire simulation cell into user specified grids and count the number of chosen atoms in each grid element. The output can be used with xfarbe directly for visualization. The output file is named `out`. The average positions of atoms can be computed if **average_positions** keyword is set to `yes`. Average positions makes sense for the surface metal atoms. The keywords specific to this compute scheme follows.

```
compute            density
element_1          O
max_bond_distance  10.0
make_grid          30 1 30
average_positions  yes
vacuum_direction   z
xfarbe_symbol      1
xfarbe_size        0.2
xfarbe_color       20
```

max_bond_distance (float)

This keyword is set if `bulk-interface` is set for **scope**. This is the distance from metal surface upto which the atoms are considered interfacial. Don't forget to set **plot** keyword.

make_grid (int int int)

This keyword allows to specify the number of grid elements in each direction. Three integer arguments for x y and z directions.

vacuum_direction (char)

This specifies the direction normal to the surface. Important for the internal workings of the program. The options are `x / y / z`.

average_positions (yes/no)

This is used only for representing the average positions of the surface atoms. In that case **element_1** must be a surface atom. The average positions are written to the file `teter.dat`. This can be read by the xfarbe program directly as the symbol file.

xfarbe_symbol (int)

This keyword is related to xfarbe. It can be set to 1,2 or 3 representing circle, square or triangle.

xfarbe_size (float)

Here the size of the **xfarbe_symbol** is set. Can use trial and error to get the right number.

xfarbe_color (int)

Check xfarbe documentation for setting this keyword.

4.2 compute density-top

This tool plots the top view showing the density of chosen atoms between the coordinates specified along the surface normal direction. The output for contour is stored in file named `out`. This plot requires average positions of surface metal atom. The same procedure as the previous compute scheme has to be used. The only difference from previous scheme is the keyword **atoms_between**, which is explained below.

```
compute            density-top
element_1          O
atoms_between
make_grid          1 1 1000
average_positions  yes
vacuum_direction   z
xfarbe_symbol      1
xfarbe_size        0.2
xfarbe_color       20
```

atoms_between (float float)

This keyword takes two values as arguments, which are the minimum and maximum values within which the atoms must be considered along the **vacuum_direction**

4.3 compute densityvelocity

This tool does the same as **compute density**, but does it for positions and velocities. Only additional keyword is **read_velocities** to be set to `yes`.

```
compute            densityvelocity
element_1          O
make_grid          1 1 1000
average_positions  yes
vacuum_direction   z
xfarbe_symbol      1
xfarbe_size        0.2
xfarbe_color       20
read_velocities    yes
```

4.4 compute rdf

This tool computes the radial pair distribution function. This requires both the keywords **element_1** and **element_2** to be set, wither to the same element symbol or two different element symbols.

```
compute            rdf
element_1          O
element_2          O
rdf_bins           100
rdf_max_rad        10.0
rdf_metal_exclude  Cu 11.8
```

rdf_bins (int)

This keyword takes an integer argument which is the number of bins for the histogram.

rdf_max_rad (float)

This keyword specifies the maximum distance upto which the distances must be correlated. The optimum value is half the smallest lattice vector.

rdf_metal_exclude (string float)

This keyword is used in case of interfaces and if you like to exclude the volume occupied by the metals especially if the metal atoms are not used for the RDF computation. The first argument is the metal species symbol and the second is the volume occupied by a single atom in $\mathring{A}^3$.

4.5 compute O-fragments

This tool counts O atom in different coordination number (species) including OH, H_2O, H_3O and O. The output is written to `O-fragments.data`. The max_bond_distance corresponds to the maximum OH bond distance for dissociation limit.

```
compute             O-fragments
max_bond_distance   1.10
```

4.6 compute percentage-coverage

This tool computes the percentage coverage of **element_1** on the surface. Some special keywords are involved. The max_bond_distance corresponds to the maximum surface-adsorbate bond distance for limiting case.

```
compute                         percentage-coverage
surface_element                 Cu
adsorbate_element               O
equilibrium_metal_bond_distance 2.53
metal_coordination_number       12 4
max_bond_distance               5.5
```

surface_element (string)

The surface metal atom symbol must be specified here.

adsorbate_element (string)

The adsorbate (interfacial) atom symbol must be specified here.

equilibrium_metal_bond_distance (float)

This special keyword is important to get the number of surface metal atoms right.

metal_coordination_number (int int)

This keyword specifies the maximum and minimum coordination for surface atoms. There is a minimum coordination number because we do not take the solvated metal ions into account.

4.7 compute diffusion-coefficient

This tool computes the diffusion coefficient of a chosen atom species. First MSD is computed and then the diffusion coefficient (D) as its continuous time derivative. The final value of D can be chosen from the converged D value from the data in file `diffco.data` or by manually fitting the MSD data in file `msd.data`.

```
compute                 diffusion-coefficient
unwrap_traj             yes
element_1               O
msd_skip_restarts       100
msd_timestep_fs         0.25
msd_lookback_ps         10
```

unwrap_traj (string)

This keyword is set to `yes` to start unwrapping the trajectory (A fortran program is used for that - will be integrated soon). As of now only works for the `orthorhombic` **cell_type**.

msd_skip_restarts (int)

Every (int)th frame will be used as a restart point for statistics.

msd_timestep_fs (float)

The timestep in femto seconds is given here.

msd_lookback_ps (float)

The total time for MSD computation is given here. If the actual trajectory is much longer, then the other frames will be used for statistical averaging.

4.8 compute VDOS

This tool finds the vibrational density of states (power spectrum) using the Fourier transformation of the velocity auto correlation functions. The velocity auto correlation function is written to `vaf.data` and the VDOS spectrum to `vdos.data`. **max_frames** is a general keyword to limit memory requirements (GPUs have meagre on board memory), but use it only for VDOS calculation, since we need every frame of the simulation to get the vibrations right.

```
compute                 VDOS
read_velocities         yes
msd_skip_restarts       1000
msd_timestep_fs         0.25
msd_lookback_ps         2
max_frames              40000
```

4.9 compute z-dist

This tool gives a 1D plot of the atomic density distribution along z direction. Vacuum direction cannot be specified for this tool. So, rotate your trajectory if needed. There are some new keywords.

```
compute                 Z-dist
element_1               O
make_grid               1 1 1000
set_max                 yes
set_maxz                40.0
set_minz                10.0
vacuum_direction        z
```

set_max (string)

If set to `yes`, they you get to specify the range within which the analysis is to be done.

set_maxz (float)

Maximum z coordinate value.

set_minz (float)

Minimum z coordinate value.

4.10 compute Hbonds

With this tool, we can count the number of hydrogen bonds along the z direction and plot them similar to the previous section. In addition, we can also plot the H-bond angles and H-bond distances. It is important to set `H` for **element_1**. **max_bond_distance** is very important since we check for **element_2** atoms within this distance from `H`. If we have two **element_2**, then we say it is a H-bond (this may not be correct). We may also have to set an angle criterium (under progress).

```
compute                 Hbonds
element_1               H
element_2               O
make_grid               1 1 1000
set_max                 yes
set_maxz                40.0
set_minz                10.0
vacuum_direction        z
max_bond_distance       2.1
```

Few more keywords are added and now it is possible to define your own Hbond criterion either based on angle/distance or a fit through OH / OHO. Both methods are coded. The basic approach in hydrogen bond identification is as follows. First, H **element_1** atoms are chosen and all O **element_2** atoms (atmost 20) within `max_bond_distance` from this atom are selected. The distances and the indices of selected **element_2** atoms are stored. Now the neighbours are sorted based on the distance from short to long. We know that the shortest OH pair is covalently bonded. Thus, the O atom here is the donor and all other in the sorted list are potential acceptors. To choose the actual acceptor, the donor acceptor (OO) distances and angles (OHO/HOO) they make with the H atom are computed and stored. Next, an optional histogram plotting to define H bond criterium is provided which can be chosen with the keyword `HB_histograms`. This can be very expensive based on the `max_bond_distance` chosen. Then, we go on finding the actual acceptor oxygen atom based on the distances, angles that we have computed earlier. Basically, this step characterizes if a H atom is Hbonded or not. There maybe situations where a H atom is H bonded to two other water molecules (this is very rare and the situation is short lived), then, we choose only one donor-acceptor. Now, the hydrogen bond information for every frame is stored. Now, we recreate the donor acceptor matrix, that will be useful in lifetime calculation. This matrix is made of 0 and 1 where 0 means no Hbond between the donor and acceptor and vice versa. Hbonds per water molecule is computed at this point and given in `HperO.data`. Now, the lifetime calculation. First, the whole trajectory is split into blocks of specified correlation time frame. Second, based on the user specified restarts, points at regular intervals from each block is chosen as a start point. At this frame, I start correlating to user defined time for those pairs of O that are Hbonded. This is the intermittent definition of H bond correlation. The auto correlation function is then fitted to a tri exponential, which gives us three lifetimes from the exponents. The shortest lifetime is the instantaneous H bond rearrangement lifetime, which is not important. The second lifetime relates to the stable H bonds without any breaking and the third one is when the recombination is allowed. It is best to choose the second lifetime as it relates to the stable H-bond lifetime where the donor-acceptor retains the Hbond. Then, derivative of this time autocorrelation function is computed in `ft.data`. The integral of auto-correlation is also written to `ct-integrate.data` and `Hbond-lifetime.data`. The average number of Hbonds and the number of Hbonds per frame is written to `Hbondnum.data` and `Avg_Hbondnum.data`.

```
HB_criterium_set       OOH-OO   #OOH-OO (distance/angle) / OHO-OH (fitted)
HB_histograms          no
max_bond_distance      3.0
max_O_O                3.5
Hbond_angle_dev        0.5235
HBlifetime             yes
HB_skip_restarts       10
HB_timestep_fs         100
HB_lookback_ps         20
```

```
HB_for                  all      #int,bulk,all
bondist_int_bulk        5.0
transient_HB_definition no
```

HB_criterium_set

This keyword takes either of the following.

- OOH-OO
- OHO-OH

If OOH-OO is used, then

4.11 compute exchange

This tool will count the number of atom exchanges between bulk and the interfacial region. This tool can also plot the exchanges along surface normal (which is experimental and not of much use as of now - needs some source code implementations). All keywords are self explanatory.

```
compute                     exchange
max_bond_distance           4.5
make_grid                   1 1 100
set_max                     yes
set_maxz                    40.0
set_minz                    10.0
vacuum_direction            z
```