RAGTAG Manual

by Kellon Belfon, Chuan Tian, James Maier, and Carlos Simmerling

To install RAGTAG

The Cuda version of **RAGTAG** works with NVIDIA GPUs. It requires the nvcc compiler for installation. The first step is to do a git clone.

```
git clone  https://github.com/kellonb/RAGTAG.git
```

The git clone command will download all the necessary files for RAGTAG. Once this is completed, change directory into RAGTAG and run

```
make install
```

Once RAGTAG is installed, the genA program can be found in the bin directory. The final step is to run the tests on your GPU to ensure RAGTAG works.

```
make test
```

Using the RAGTAG input file generator

Introduction

The RAGTAG input file generator is a python script, create_genA_input.py. To use this script you will need the genA.py and pytraj python modules. You can download everything related to RAGTAG from my github (https://github.com/kellonb/RAGTAG) and everything related to pytraj here (https://amber-md.github.io/pytraj/latest/index.html). The create_genA_input.py and genA.py module are located in the ***scripts/Input_generator*** directory of RAGTAG. The function of the input generator is to create an input file for the Cuda version of the RAGTAG program. The input file format is shown below and an example is provided. A user has the option to create the file by hand. If the input generator is used, a file named genA_input will be created in the user's directory. This file is further used to run RAGTAG.

The file format of the input

The RAGTAG input file has a unique format. You can use these examples to ensure that the input file generator produces a file with the right format. Please feel free to email me (kellonbelfon@gmail.com) if there are any issues. Additionally, there are examples of different input files in the ***test_cases*** directory. We choose this format to ensure RAGTAG works with any MD packages. All a user needs is a set of conformations, the dihedral values for each conformation, and the QM and MM0 energies.

The general input file format is :

```
-<dihedral 1> <AMBER atom types for dihedral 1>   -<fitting group index> <periodicity of cosines>
-<dihedral 2> <AMBER atom types for dihedral 2>    -<fitting group index> <periodicity of cosines>
<name of data set1> <<weight> <dihedral 1> <dihedral 2>  ><number of dihedrals in data set>
 <dihedral 1 value> <dihedral 2 value> <E_QM> <E_MM0>
 <dihedral 1 value> <dihedral 2 value> <E_QM> <E_MM0>
                ...
/
<name of data set2> <<weight> <dihedral 1> <dihedral 2>  ><number of dihedrals in data set>
 <dihedral 1 value> <dihedral 2 value> <E_QM> <E_MM0>
```

<dihedral 1 value> <dihedral 2 value> <E_QM> <E_MM0>

...

/

Caveats to the input file

The periodicity must be listed from largest to smallest e.g 4 3 2 1 or 3 1 or 4 2 1
The fitting groups must be in order. So the first dataset must have the first set of fitting groups.
The dihedral columns must be written in the order of the fitting group.

Description of the file format:

<dihedral> is the name of the dihedral being fitted e.g phi, psi, chi1, chi2, chi3, etc
<AMBER atom type for dihedral 1> e.g chi1 is N -CX-2C-2C for Met, get this from a frcmod file
<name of data set> is any name, e.g Metalpha, Metbeta, Metcharge
<dihedral 1 value> this is the dihedral value (deg) of the optimized QM structures e.g 105.62
<E_QM> the QM energy of conformation i with dihedral restrained to the value
<E_MM> the MM energy of conformation i with dihedral restrained to the value, the target dihedral parameters are zeroed in the frcmod file
... repeat for all conformations within a break
/ (refer to as break (brk))
a break separate conformations that are different e.g alpha backbone, beta backbone, charge amino acids
<weight> weight of the AAE of a dataset. It is 2/(nconf)*(nconf-1) * weight_value_used. It can be used to apply a weight to a dataset over the others. This can be useful if you want to use high energy structures. You can include them as a separate dataset and add a weight to them so they don't count as much to the final score.
<number of dihedrals in data set> This is the total number of dihedrals in the dataset.

An example of an input file is:

```
-chi1 N- CX-2C-2C -0 4 2 1
-chi1p 2C-2C-CX-C  -0 4 2 1
-chi2 CX-2C-2C-SE -1 4 3 2 1
-chi3 2C-2C-SE-CT -2 4 2 2 1
MSEalpha <1 chi1 chi1p chi2 chi3 >4
 -60.0000 60.0000 -151.3518 -172.2707 -1865195.7795 -20.7497
 60.0000 120.0000 -133.4966  162.7864 -1865192.2533 -17.6159
 -20.2001 100.0001 -175.8127 -19.7117 -1865192.8184 -18.2698
 -40.0002 79.9991 -141.6441 -78.1196 -1865195.3088 -20.3646
 -32.3320 88.3321 -156.1539 -74.6189 -1865195.4806 -20.3748
 103.2203 -42.9881 -160.1677 -44.8386 -1865195.7975 -20.6859
 -34.0201 86.0210 -163.3005 -151.3139 -1865195.9084 -20.6702
 -20.2021 100.2011 -170.0999 140.3563 -1865196.3772 -21.3357
 -20.1010  100.1021 105.7171 71.0025 -1865194.7143 -19.1208
 66.9291   126.9300 61.0493  -146.3632 -1865193.9557 -17.8761
/
MSEopt <1 chi1 chi1p chi2 chi3 >4
 -43.2101 83.1021 107.8337 -168.6532 -1865197.6927 -21.6032
 129.1910 -70.1023 -153.5202 72.4044 -1865200.9804 -25.0604
```

```
 -34.5696 102.2111 151.9807 -57.4394 -1865201.4222 -25.4324
 -45.2929 85.0000 167.0896 133.5922 -1865201.2247 -24.6531
 30.3003 150.3021 -170.6856 -74.6871 -1865202.0865 -26.1129
 170.2901 -110.2014 169.0200 -162.0158 -1865201.5012 -25.5234
 -30.2011 90.2011 153.6404 -177.6957 -1865200.5430 -24.6895
/
```

In the example above, we have two datasets. One dataset (MSEalpha) has 10 structures with the dipeptide backbone restrain to alpha (phi-60,psi40) and the other dataset (MSEopt) has 7 structures with the dipeptide backbone restain to beta (phi-135,psi135). We are fitting 4 dihedrals (nDih = 4) but only using 3 fitting group (nFg = 3). Chi1 and Chi1p are fitted together and hence are in the same fitting group (fitting group 0). This means that the same set of parameters will be used for both chi1 and chi1p. Chi2 is fitting group 1 and chi3 is fitting group 2. For chi2 and chi3 we are fitting 4 periodicities or 4 cosines (n-4, n=3, n=2, n=1). For chi1 we are fitting only n=4, n=2, and n=1. This will zero the amplitude when n=3.

Another example:

```
-chi1 N -CX-2C-2C -0  4 3 2 1
-op CX-3C-OZ-P   -1  4 2
-po1 3C-OZ-P -OX   -2  3
-po2 3C-OZ-P -OX   -2  3
-po3 3C-OZ-P -OX   -2  3
-chip 2C-2C-CX-C  -3 4 2
-chi2 CX-2C-2C-SE  -4 4 3 1
-chi3 2C-2C-SE-CT  -5 4 3 2 1
TPalpha <1 chi1 op po1 po2 po3 >5
   63.0874   -59.0409    90.5488  -150.3919   -30.1205    -736778.2909  -149.7037
  -36.7006   104.9759   170.1583   -70.1912    49.8179    -736770.0137  -155.9444
  -36.8163   105.3888    49.7693   170.0945   -70.2459    -736770.0241  -156.1593
  -36.8556   105.3410   170.1453   -70.1966    49.8170    -736770.0166  -156.1419
  -36.6658   104.8297    49.7303   170.0774   -70.2778    -736770.0295  -155.8546
  -37.5104   107.2896    49.5865   169.8546   -70.4529    -736770.0479  -156.8462
  -36.9169   105.5614   170.1291   -70.2077    49.8067    -736770.0348  -156.2559
  -36.8539   105.4027   170.1581   -70.1815    49.8325    -736770.0027  -156.1772
/
TPopt <1 chi1 op po1 po2 po3 >5
  -24.3298   101.6504    64.3936  -176.1215   -55.8130    -736741.5425  -131.9036
  175.5764    75.4107   -69.0075    51.7490   171.6296    -736760.3022  -149.8228
  176.0763    74.2118   172.0751   -68.6411    52.1404    -736760.2589  -148.9124
 -139.5829   -52.5569   -49.5455    71.5395  -169.4028    -736756.7683  -137.7741
 -140.3095   -51.9426   -50.1678    71.1538  -170.0265    -736756.7485  -136.7837
  175.8465    74.6645   171.8951   -68.7902    51.9834    -736760.2935  -149.2883
  176.0025    74.4861    52.0473   171.9681   -68.7286    -736760.2824  -149.1695
  175.8982    74.7263   -68.8178    51.9554   171.8641    -736760.2382  -149.2873
 -140.7120   -51.6848   -50.4853    70.9368  -170.3485    -736756.7682  -136.3393
  -24.2830   101.4697    64.3229  -176.1973   -55.8851    -736741.5547  -131.8515
  175.8496    74.7307    51.9476   171.8457   -68.8324    -736760.2925  -149.2969
  176.2116    73.8869   -68.5078    52.2761   172.2336    -736760.2629  -148.6687
/
```

```
MSEalpha <1 chi1 chip chi2 chi3 >4
 -60.0000  120.000 -151.3518 -172.2707 -1865195.7795 -20.7497
 +60.0000  180.000  -133.4966 +162.7864 -1865192.2533 -17.6159
 -20.2001  100.2001 -175.8127 -19.7117 -1865192.8184 -18.2698
 -40.0002   80.0002 -141.6441 -78.1196 -1865195.3088 -20.3646
 -32.3320   88.3320  -74.6189  80.1439 -1865195.4806 -20.3748
 +103.2203 -43.2203 -160.1677 -44.8386 -1865195.7975 -20.6859
 -34.0201   86.0201 -163.3005 -151.3139 -1865195.9084 -20.6702
 -20.2021   100.2001 -170.0999 +140.3563 -1865196.3772 -21.3357
 -20.1010   100.1010 +105.7171 +71.0025 -1865194.7143 -19.1208
 +66.9291   -6.9291  +61.0493  -146.3632 -1865193.9557 -17.8761
/
MSEopt <1 chi1 chip chi2 chi3 >4
 -43.2101  77.2101 +107.8337 -168.6532 -1865197.6927 -21.6032
 +165.1010 -105.1011 +159.6474 -95.6575 -1865199.4515 -23.8997
 +135.1019 -75.1019 +148.1114 +30.8011 -1865197.5170 -21.9696
 +129.1910 -69.1910 -153.5202 +72.4044 -1865200.9804 -25.0604
 -34.5696   85.4304 +151.9807 -57.4394 -1865201.4222 -25.4324
 -45.2929   75.2929 +167.0896 +133.5922 -1865201.2247 -24.6531
 +30.3003 150.30003 -170.6856 -74.6871 -1865202.0865 -26.1129
 +170.2901 -110.2901 +169.0200 -162.0158 -1865201.5012 -25.5234
 -30.2011 89.7989 +153.6404 -177.6957 -1865200.5430 -24.6895
 +102.2020 -42.2020 +164.0768 -136.6440 -1865200.7063 -24.8092
/
```

In the example above, we have four datasets. MSE has the same chi1 with TP so they are fitted together. Note that the order of the fitting group matters. In this case the dihedral columns are always listed in the order of the fitting groups. So therefore dihedrals for fitting group 2 (column 3,4,5 in TPalpha) should not come before fitting group 0.

Flags for the input generator

The input generator uses three flags that direct to three files.
-i flag specifies the **input file**, which has a few namelist described below
-at flag specifies the **dihedral information file**
-top flag specifies the **topology information file**. Only use this flag if nbreaks > 1

Usage

If nbreaks = 1:
python create_genA_input.py -i input -at dihedral_info
If nbreaks > 1:
python create_genA_input.py -i input -at dihedral_info -top top_info

Files needed for the input generator

Input file:

The following namelist options are specific to the input file. They follow the amber format of namelist = option, . Do not forget to put the comma after

top This variable can be used to specify the topology name and/or the path to the topology file. The topology file is used to calculate the dihedrals for the RAGTAG program using the coordinates of MM structures. It is also used to write a trajectory of the conformations used for the fitting to be visualized in VMD or Chimera. If *nbreaks* > 1, do not use the *top* variable in the input file. It will be read from the topology information file.

mol_name This variable can be used to specify the molecule name. For example, butane. When there is only 1 dataset (nbreaks = 1), mol_name is used as the name of the trajectory file written for visualization, and as the dataset name in the RAGTAG input file. If *nbreaks* > 1, do not use the *mol_name* variable in the input file. It will be read from the topology information file.

nconf This variable can be used to specify the number of conformations that are being used in the fitting process. If *nbreaks* > 1, do not use the *nconf* variable in the input file. It will be read from the topology information file. This is important because a user may have different number of conformations for their various datasets.

nbreaks This variable can be used to specify the number of datasets. Different datasets are used when fitting different molecules with similar dihedrals. For example Methionine and Glutamine share the same atom types for chi1. Therefore one can fit both molecules at the same time by separating them into two different dataset.

dihnum This variable can be used to specify the number of dihedrals being fit.

flag_mm_read This variable acts as a flag to change the way the MM energy is extracted.
Flag options:
0: get MM energy using the mol_name. The structures have to be of format mol_name.000.energy to mol_name.nconf.energy (e.g butane000.energy to 1butane012.energy when *nconf* = 12
mol_name.nconf (mol_name.012)
1: get energy from a filename where each line has a path to an MM file with MM energy
2: get the energies from one file with format molecule_index MM energy (kcal/mol)

flag_qm_read This variable acts as a flag to change the way the QM energies is extracted.
Flag options:
0: get energy from an orca relaxed scan
1: get energy from a filename where each line has a path to an orca.log file with QM energy
2: get the energies from one file with format molecule_index Qm Energy, but QM energy is in hartrees
3: get the energies from one file with format molecule_index Qm Energy, but QM energy is kcal/mol

mm_read_file This variable can be used to specify the filename that controls how the MM energy is extracted. If *flag_mm_read* = 1, each line in the file is just a path to the .info or .out file. The input generator reads the .info file by default. If *flag_mm_read* = 2, then the file will have the molecule index in the first column and the energies in kcal/mol in the second

column for each conformation. In the mm_read_file if nbreaks > 1 then separate each dataset with the word break. See an example below.

qm_read_file This variable can be used to specify the filename that controls how the QM energy is extracted. If *flag_qm_read* = 1, each line in the file is just a path to the orca log file. currently the input generator only support reading from orca log files when *flag_qm_read* = 1. If you use another QM package such as Gaussian, you can extract the energy and use *flag_qm_read* = 2 instead. If *flag_qm_read* = 2 or 3, then the file will have the molecule index in the first column and the energies in hartrees (*flag_qm_read* =2) or kcal/mol (*flag_qm_read* = 3) in the second column for each conformation. In the *qm_read_file* if *nbreaks* > 1 then separate each dataset with the word **break**. See an example below.

Example of input files:

Example of an input file when *nbreaks* = 1:

```
top = ../03.mm0/butane_0dih.parm7, mol_name = ../03.mm0/butane_mm0_, nconf = 12,
dihnum = 1, nbreaks=1,
flag_mm_read = 0,
flag_qm_read = 0, qm_read_file = ../02.qmorca/orca.log
```

Example of an input file when *nbreaks* > 1:

```
dihnum = 3, nbreaks = 2,
flag_mm_read = 1, mm_read_file = ./list_mmstructure.dat,
flag_qm_read = 1, qm_read_file = ./list_qmstructure.dat,
```

Example of a *qm_read_file* when *flag_qm_read* = 1 :

```
../alpha/struct0/MSE.log
../alpha/struct1/MSE.log
../alpha/struct2/MSE.log
break
../opt/struct0/MSE.log
../opt/struct1/MSE.log
../opt/struct2/MSE.log
```

Example of a *qm_read_file* when *flag_qm_read* = 2 :

```
  1.00000000   -157.96305502
  2.00000000   -157.96053810
  3.00000000   -157.95765966
  4.00000000   -157.96026532
  5.00000000   -157.96209966
  6.00000000   -157.95817518
```

Dihedral info file:

The dihedral info file has three columns.
The first column contains the dihedral name. For example chi1.
The second column contains the atom mask in amber format. For example @2 @3 @4 @6. The mask is used to calculate the dihedral value for each conformation within a dataset.
The third column is the atom type. For example N -CX-2C-2C
The fourth column is the fitting group index. For example 0 or 1 or 2. Fitting groups are dihedrals that share the same atom types or the same dihedral.
The fifth column is the periodicity that you want to fit. For example 3 2 if you want to fit for n=3 and n=2.

Each line will represent a given dihedral and each entry is separated by a comma. If you have multiple datasets with the same dihedral name, mask and atomtype, still repeat it. Look at examples before.

Example for 2 dihedrals with 1 dataset (nbreaks = 1)

```
#name      mask                    atomtype   fitting group   periodicity
chi1,    @7 @9 @11 @14,     N -CX-2C-2C,          0,          4 2 1
chi2,    @9 @11 @14 @17,    CX-2C-2C-SE,          1,          4 3 2 1
```

Example for 3 dihedrals with 2 dataset (nbreaks = 2) The two data sets is different backbone conformation

```
#name      mask                    atomtype   fitting group   periodicity
chi1,    @7 @9 @11 @14,     N -CX-2C-2C,          0,          4 2 1
chi2,    @9 @11 @14 @17,    CX-2C-2C-SE,          1,          4 3 2 1
chi3,    @11 @14 @17 @18,   2C-2C-SE-CT,          2,          4 3 2 1
chi1,    @7 @9 @11 @14,     N -CX-2C-2C,          0,          4 2 1
chi2,    @9 @11 @14 @17,    CX-2C-2C-SE,          1,          4 3 2 1
chi3,    @11 @14 @17 @18,   2C-2C-SE-CT,          2,          4 3 2 1
```

Example for 1 dihedral with 2 dataset (nbreaks = 2)

```
#name      mask                    atomtype   fitting group   periodicity
chi1,    @7 @9 @11 @14,     N -CX-2C-2C,          0,          4 2 1
chi1,    @7 @9 @12 @15,     N -CX-2C-2C,          0,          4 2 1
```

Topology info file:

The topology info file is only needed when you have multiple datasets (nbreaks > 1).

The first column is the topology file name for a specific dataset.
The second column is the name of a specific dataset
The third column is the number of conformations (nconf) in a specific dataset
The fourth column is if you want to weigh a dataset by some value. That value is divided by(nconf * ((nconf-1)/2)), where nconf is the number of conformations in that given dataset. In our manuscripts, we used a weight of 1 for all datasets. It can be advantageous if you want to weigh a dataset with high energy conformations as shown in the RAGTAG manuscript. This column is optional and if not specified a weight of 1 will be used making the equation 1 / (nconf * ((nconf-1)/2)) == 2/(nconf)*(nconf-1).

Example for 3 dihedrals with 2 dataset (nbreaks = 2) The two datasets is different backbone conformation

```
# topology                  datasetname      nconf
```

```
MSE_ff14SB_0dih.parm7,    MSEalpha,          7
MSE_ff14SB_0dih.parm7,    MSEopt,            9
```

Example for 3 dihedrals with 2 dataset (nbreaks = 2) with some arbitrary weights, The two datasets is different backbone conformation

```
# topology                datasetname    nconf    weight
MSE_ff14SB_0dih.parm7,    MSEalpha,      6        1
MSE_ff14SB_0dih.parm7,    MSEopt,        6        1
MSE_ff14SB_0dih.parm7,    MSEhigheng     4        0.01
```

Example for 3 dihedrals with 2 dataset (nbreaks = 2) The two datasets is different amino acids

```
# topology                datasetname    nconf
MET_ff14SB_0dih.parm7,    MET,           7
GLU_ff14SB_0dih.parm7,    GLU,           9
```

Example for 1 dihedral with 3 dataset (nbreaks = 2) The three datasets is different amino acids

```
# topology                datasetname    nconf
MET_ff14SB_0dih.parm7,    MET,           7
GLU_ff14SB_0dih.parm7,    GLU,           9
GLN_ff14SB_0dih.parm7,    GLN,           6
```

Using the RAGTAG program

Introduction

RAGTAG is a genetic algorithm used to optimize dihedral parameter for Amber force field using graphic processing unit (gpu). The program optimizes the difference between QM energies and MM energies of a particular force field for a given dihedral using an objective function shown below. RAGTAG is advantageous when your parameter search space is discrete. A cpu version of RAGTAG was written and used by James Maier to obtain the dihedral parameters for the side chains in ff14SB. Following the publication of ff14SB, James Maier wrote a GPU version of the code referring to RAGTAG_v1. Following that version, Kellon Belfon and Chuan Tian have contributed to the code to create RAGTAG_v2. All of this work was done in Prof. Carlos Simmerling's lab at Stony Brook University.

Functionality

- Can fit any number or any combination of cosines
- Can use random amplitude parameters or existing amplitude parameters as initial guess
- Only fit the amplitude parameters, The amplitude describes the energy barrier height of a given cosine with n periodicity. In this version of RAGTAG, the phase parameter is dependent on the sign of the amplitude. If the amplitude is negative, then there is a phase shift of 180 degrees and if the amplitude is positive then a phase shift of zero is applied. Using a binary approach to the phase parameters allowed the code to be faster, since we can pre-calculate the cosine term in the dihedral energy equation and store them in an array (*tset)* on the GPU.

Flags used for RAGTAG

RAGTAG uses 8 flags that specifies for 8 files.

-i flag specifies the **input file**, which is the file (genA_input) created by the input generator.

-p flag specifies the **parameter file**, which is the file with the genetic operators name list option.

-r flag specifies for the option to save a **restart file**. The restart file has the set of amplitudes that can be used to restart the optimization if the results are not the best. Restarting the optimization can be useful to change the genetic operators in the parameter file. For example increasing the mutation rate.

-s flag specifies the option to save the scores for a given number of chromosomes for a given number of generation. This can be used for plotting the convergence of your score using the plot_scores.py python script.

-f flag specifies the option to print out an amber frcmod file, ready to be used by tleap to incorporate the new dihedral parameters using tleap *loadamberparams* function.

-y flag specifies the option to print information on the fit. This file is great to analyze how good is the parameters generated from the program. Additionally, there are two python scripts (cal_ree_init.py and cal_ree_final.py) that can be useful in that regard.

-o flags specify the option to save a log file.

-c flags specify the option to load a restart file. This flag can also be used to give the program a set of previous amber amplitude parameters. The python script write_restart_RAGTAG.py can be used to extract the amplitudes from an amber frcmod file and create a file that has the amplitudes as an input to RAGTAG. The amber frcmod file should only have the dihedrals that you are fitting and it dihedrals should be in the order that you are fitting. For example, if I am fitting two dihedrals chi2 and chi3 and my RAGTAG_input file has the order of chi2 column before chi3 column, then my dihedral.frcmod file will have chi2 amber parameters first, then chi3 amber parameters. The command is:

python write_restart.py dihedral.frcmod [nchrom]

nchrom is an option of how much chromosomes of the previous amber parameters do you want to start RAGTAG with. For example, if you do not want the previous amber parameters to compete with Random parameters then nchrom will be the same as pSize (see below for a description of pSize). If You want. The previous amber parameters to compete with random generated parameters then *nchrom* can be any number between 1 and pSize.

Files needed for RAGTAG

The input file and the parameter file are needed for the program to run. The restart file can be used for restarting the genetic algorithm or for loading previous amber parameters. The other files are mostly different output file.

Input file:

The input file is the file generated by the input generator described above. An example can be found in the second page of the manual.

Parameter file:

The following namelist options are specific to the parameter file. They follow the amber format of namelist = option but instead only listed line by line.

pSize This variable can be used to specify the population size. The population size is the number of Chromosomes in a population. Too small of population size limits the search space and too

large of a population can slow the algorithm done. An optimal population size is also dependent on the GPU used to run the calculations. There is a summary of the CUDA cores for the popular GPUs below. The program uses a block size of 256 threads to run the calculations on a given chromosome. Each thread is responsible for the instructions to operate on a given chromosome. Based on the CUDA cores optimal *pSize* is 1500-2000. Larger psize can be used, but some slowing down will be expected. On GPUs such as the 1080Ti, psize of up to 3500 can be used and will give the same gpu time as on a 980 with a psize of 2000.

nGen This variable can be used to specify the number of generations that the genetic algorithm will cycle. Once *ngen* is completed the genetic algorithm will stop. It is safe to start with a nGen greater than 1000. Convergence can be checked by looking to see if you have a dominant chromosome in your population from the log file. The larger *ngen* is the slower the code.

pMut This variable controls the probability of mutation.

max This variable controls the maximal permissible mutation. A mutation is a small perturbation to the genes (amplitudes) on a given chromosome. Selection of a chromosome for mutation is based on a logic condition, where if a random number is less than the input parameter *pMut* then the amplitude is perturbed by the equation below, where V is the amplitude.

V + (r * (2 * (max/pMut) - max))

Mutations is a useful tool to navigate small changes in the parameter search space.

pCross This variable can be used to change the crossover rate. A crossover between two selected parents occurs during mating. Genes (amplitude parameters) are swapped at a given cross-over point on a chromosome using the equation below. A crossover rate of 0.8 is ideal.

$$\text{crosspt} = 1 + \frac{random\ \#}{pcross * (genomesize - 1)}$$

In the example below, for a genome size of 8 and a pcross number of 0.8, a crosspt (which is always an integer) of 3 was calculated (shown in blue). With this crosspt one offspring will have the first three genes (amplitude) of parent1 and the last 5 genes of parent 2 and the other will have the first three genes of parent 2 and the last 5 genes of parent 1. Crossover is useful to navigate large jumps on the parameter search space, since it creates new chromosomes with entirely different fitness.

Parent 1

$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{4,2}$	$V_{3,2}$	$V_{2,2}$	$V_{1,2}$

Parent 2

$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{4,2}$	$V_{3,2}$	$V_{2,2}$	$V_{1,2}$

Offspring 1

$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{4,2}$	$V_{3,2}$	$V_{2,2}$	$V_{1,2}$

Offspring 2

$V_{4,1}$	$V_{3,1}$	$V_{2,1}$	$V_{1,1}$	$V_{4,2}$	$V_{3,2}$	$V_{2,2}$	$V_{1,2}$

rseed This variable is a 6 digit number that is used as a random seed when generating the random set of numbers.

peng This variable can be used to print scores every n generations. It is useful to obtain scores over generations to plot for convergence. The scores are printed to the score file.

ncp This variable can be used to print the number of chromosome scores. When specified the top ncp chromosomes scores and areas are printed to the score file.

nCos This variable specifies the number of cosine to be used for the fitting of the dihedrals. If a user only want to use 3 cosines then that option is available. Additional if *nCos* and *aZp* is set then a user can specify which periodicity to be used with *nCos. nCos is deprecated.*

keep This variable can be used to specify how much of the previous generation to keep during the elitist regime. A default of 20 % of the previous generation best chromosome are kept to compete with the offspring in the current generation.

nDataset This variable specify the number of datasets you have for fitting. The default is 1.

nChrom The number of chromosomes with old forcefield parameters. You can make the old forcefield parameters compete against random or only start with old forcefield parameters/initial guest/previous

nFg The number of fitting groups.
nDih The number of dihedrals

Frcmod file:

An amber frcmod file is created once RAGTAG is completed. The frcmod file is specific to amber and able to add the newly generated parameter to your simulation without changing the main parameter file in amber. An example of a frcmod file is shown below.

Example of a frcmod file generated by RAGTAG for phi and psi dihedrals of dialanine using 4 cosines

```
frcmod from RAGTAG.cu
DIHE
C -N -CX-C      1   0.742034     0    -1
C -N -CX-C      1   0.096635     0    -2
C -N -CX-C      1   0.197728     0    -3
C -N -CX-C      1   0.185530   180     4
N -CX-C -N      1   0.188103   180    -1
N -CX-C -N      1   1.107180   180    -2
N -CX-C -N      1   0.502601   180    -3
N -CX-C -N      1   0.030636   180     4
```

Score file:

The score file is created once ncp and peng is selected. It contains the score and probability of selection for mating (areas) of a given chromosome. The generation of -1 is the initial scores of ncp chromosomes. Those initial scores are either from random numbers or from loaded amber parameters. The score is the Average Absolute energy and is calculated using equation4 in RAGTAG manuscript. A score of less than 2 kcal/mol is desirable.

Example of a score file generated by RAGTAG for peng = 5 and ncp = 4

#Generation	Chromosomes	Scores	areas (probability of mating)
-1	0	1.68559	
-1	1	1.73389	
-1	2	1.77776	
-1	3	1.78285	
0	0	1.68374	0.367879
0	1	1.68559	0.367879
0	2	1.70103	0.367879
0	3	1.73389	0.190115
5	0	1.59363	0.367879
5	1	1.60766	0.367879
5	2	1.61304	0.367879
5	3	1.61568	0.211045
5	4	1.62401	0.29047

Fit file:

The fit file is an output file that describes the accuracy of the fitting procedure. The first column is the initial dE. This list of energy is the target energy that the genetic algorithm tries to reproduce in pairwise. dE is calculated by using the first conformation in each dataset as the reference structure. Following that for each conformation dE is Eqm,conf - Eqm,ref - Emm,conf - Emm,ref. As a result, the first conformation in each dataset is zero. However when the score is calculated to evaluate how fit a chromosome dE + Edih is calculated pairwise. Regardless of a perfect fit, an AAE = 0 will have the second column which is the dE after the fit being exactly similar to the initial dE in the first column. Therefore it is fitting that the third column describes the error in the fitting. This file is useful to get a good view of the accuracy of the fit. The fit file also has the pairwise REE for

each pair and also the individual dataset AAE. Remember RAGTAG score is an average of the AAE for each dataset.

Log file:

The logfile repeats the genetic operator parameters in the beginning. Then it is followed by the score and dihedral parameters for each chromosome at the end of the genetic algorithm. To the bottom of the logfile is the time (millisec) it took for the algorithm to find dihedral parameters.

Key definitions:

genomeSize This describes the size of the chromosome. It is calculated by multiplying the number of dihedral being fitted by the number of cosines. For example if a 2 dihedrals are being fit with 4 cosine functions then the genomeSize will be 8. Therefore a given chromosome will have 8 genes (amplitudes). Each chromosome is then stored in the Vs array which has dimension pSize * genomeSize.

tset This describes the precalculated cosine terms which are cos (periodicity * dih). The dihedrals are in radians. Therefore each gene has its corresponding tset. In otherwise, tset has the same dimension as Vs so we can multiply the Vs array by the tset array to produce a shorter form of the amber dihedral energy. By doing this for a given gene in a given chromosome V * cos (periodicity * dih) is evaluated to obtain the dihedral energy which is summed and added to dE.

tgts An array that hold dE.

areas Selection of parents for mating was done using the fitness proportionate selection algorithm as a genetic operator. A fitness is applied to each chromosome using the scoring function described below. Each chromosome occupy an area based on it score; the larger the score, the larger the area it occupies. The area of a chromosome is calculated using the equation below, where scorei is the score of a given chromosome, and scoreb is the score of the best scoring chromosome. Evaluation of equation x shows that a chromosome with a more fit score will have a larger area and hence a greater probability of being selected for mating.

$$\text{Areas}_i = e^{-scorei\ /\ scoreb}$$

A random number is generated and determines how far to the right you will select individuals.
The fittest individual has the greatest probability of being chosen since they area is larger (so they are longer||). The larger the area or longer the space then the greater the probability of that individual being hit by a dart (selected for mating based on a random number.

kernel This is a small program or function that is launch on a GPU. We have kernels that operate on one chromosome for each stage of the genetic algorithm (scoring, mating, sorting, and mutation). Therefore the parallelization that is achieved here is that each chromosome can be scored, mate, mutate and sorted in parallel instead of in serial as previously done on the cpu.

Description of the GPU architecture:

I am not an expert on GPU architecture but this is my interpretation. The higher part of the hierarchy is a grid. A grid contains a set of blocks. A block contains a set of threads. Threads are mapped to cuda cores. In general, the maximum number of threads a block can have is 516. In our implementation of RAGTAG, each block contains a maximum of 256 threads. Once a thread block is added to a streaming multiprocessor, it is divided into sets of 32 threads called warps. Threads in the same warp can communicate with each other and share the same memory, barrier synchronization, follow the same instructions, and run on the same stream processor. Each thread contains a particular set of instructions (genetic operations) that are used to index into a given chromosome. Therefore you can think about it, as a chromosome being mapped to a thread. For example, if pSize is 2000 then RAGTAG will need 2000 threads to run.

GTX	CUDA Cores	SM
680	1536	8
780	2304	12
780Ti	2880	15
970	1664	
980	2048	
1080	2560	
1080Ti	3584	
2080Ti	4350	

Nblocks, Blocksize,
Blocksize = 256
Nblocks= psize+blocksize-1 / blocksize
Max psize of 2000
N blocks = 2000 + 255 / 256 = 8 blocks
Max = 8 * 256 = 2048 cores

Fermi: 16SM (streaming multiprocessor)
1 SM contains 32 cuda cores, 2 warp scheduler and dispatch unit