GPUGA:
Graphics Processing Units Genetic Algorithm

Reference Manual

Version 1.0

(Nov 06, 2019)

Author:
Zheyong Fan (Bohai University and Aalto University)

Contents

Chapter 1

Introduction

1.1 What is GPUGA?

GPUGA stands for Graphics Processing Units Genetic Algorithm. It is a highly efficient empirical potential fitting code using the genetic algorithm (GA) implemented on graphics processing units (GPU). The implementation language is CUDA C++. **Fitting one empirical potential only takes about one minute using GPUGA.**

GPUGA was developed for Fan et al. (2019). If you use GPUGA in your published work, we kindly ask you to cite this paper.

This is version 1.0, and it can only be used to fit the minimal Tersoff potential as proposed in Fan et al. (2019). We aim to implement more empirical potentials for version 2.0.

All the potentials considered in GPUGA will be implemented in the GPUMD package, which can be found here:

https://github.com/brucefan1983

1.2 Feedbacks

You can email Dr. Zheyong Fan if you find errors in the manual or bugs in the source code, or have any suggestions/questions about the manual and code. The following email address can be used:

- brucenju(at)gmail.com

1.3 Acknowledgments

We acknowledge the computational resources provided by Aalto Science-IT project and Finland's IT Center for Science (CSC).

Chapter 2

Theoretical formalisms

See Fan et al. (2019).

Chapter 3

Using GPUGA

The code has only been tested in linux operating systems and we assume that the user is using a linux operating system to compile and run this code.

3.1 Compile the code and run the examples

3.1.1 Compiling

After downloading and unpacking `GPUGA`, one can see four folders: `src`, `examples`, `tools`, and `doc`. The folder `src` contains all the source files. The folder `examples` contains all the examples. The folder `tools` contains a Matlab script for plotting Fig. 3 in Fan et al. (2019). The folder `doc` contains the pdf file you are reading and the source files generating it.

To compile the code, go to the `src` folder and type

```
make
```

in the command line. This will produce an executable called `gpuga` in the `src` folder. The second line of `makefile` reads

```
CFLAGS = -std=c++11 -O3 -arch=sm_50 --use_fast_math
```

Change `50` to the appropriate "compute capability" of your GPU. The minimum compute capability supported by `GPUGA` is 3.5.

3.1.2 Running

Go to the directory where you can see the `src` folder and type

```
src/gpuga < examples/input.txt
```

This will run the prepared example. To run your own calculations, just replace the folder name (including absolute or relative path) as specified in `examples/input.txt` to your own working directory. Then you need to prepare some input files in the working directory, as described below.

3.2 Input files for GPUGA

To run one simulation with `GPUGA`, you need to prepare three input files in your chosen working directory: `ga.in`, `potential.in`, and `train.in`. We describe them below.

3.2.1 The ga.in input file

This file contains the controlling parameters defining the GA evolution. In this input file, blank lines and lines starting with # are ignored. Each non-empty line starts with a keyword followed by one parameter. The valid keywords and their parameters are listed below.

1. `maximum_generation`
   This keyword needs one parameter, which is the maximum number of generations in the GA evolution. It should be a positive integer. The default value is 1000.

2. `population_size`
   This keyword needs one parameter, which is the population size (number of individuals in one generation). It should be no less than 20 and should be a multiple of 10. The default value is 200.

3. `parent_number`
   This keyword needs one parameter, which is the number of parents in one generation. It can be no less than 10 and should be a multiple of 10. The default value is 100.

4. `mutation_rate`
   This keyword needs one parameter, which is the initial mutation rate in the GA evolution. It should be in $[0, 1]$. The default value is 0.2. The mutation rate will linearly decrease and reach 0 up to step `maximum_generation`.

3.2.2 The potential.in input file

This file contains information about the potential to be fitted. It has a fixed format. For example, the file `examples/si_diamond/potential.in` reads:

```
potential_type 1
cutoff         3.0
weight_force   0.05
weight_energy  0.15
weight_virial  0.8
D0             2.9 3.3
alpha          1.3 1.5
r0             2.2 2.4
S              2 2
n              0.5 0.8
beta           0 0.4
h              -0.8 -0.6
R1             2.8 2.8
R2             3.2 3.2
```

At each line, there is one character string and one or two numbers. To do new experiments, just keep the strings unchanged and modify the numbers. Here are some explanations:

- Line 1: The type of the potential to be fitted, which can only be 1 (means the minimal Tersoff potential Fan et al. (2019)) in this version.

- Line 2: The cutoff distance used for building the neighbor list for each configuration (see below), which should be a positive number.
- Lines 3-5: The weighting factors for force, energy, and virial. They should be non-negative numbers. It is good to make their sum to be 1, although the code does not complain if this is not the case.
- Lines 6-14: The lower (the first number in each line) and upper (the second number in each line) bounds of the potential parameters for the minimal Tersoff potential. The order of the parameters are the same as those in Table II of Fan et al. (2019).

3.2.3 The train.in input file

This file contains all the training data, possibly from DFT calculations. The format is fixed:

```
Nc Nc_force
N_1
N_2
...
N_Nc
Data for force configuration 1
Data for force configuration 2
...
Data for force configuration Nc_force
Data for energy/virial configuration 1
Data for energy/virial configuration 2
...
Data for energy/virial configuration Nc - Nc_force
```

Here,

- `Nc` is the total number of configurations.
- `Nc_force` is the total number of force configurations. `Nc - Nc_force` is the number of energy/virial configurations.
- `N_i` is the number of atoms in configuration `i`. All the force configurations should come before all the energy/virial configurations.
- Data for one force configuration occupy `N_i + 1` lines. The first line should have nine numbers defining the cell vectors ($\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$):

  ```
  ax ay az bx by bz cx cy cz
  ```

  In the remaining `N_i` lines, each line contains 7 numbers, corresponding to the atom type (not used in this version and can be set to 0), position components, and force components:

  ```
  type x y z fx fy fz
  ```

- Data for one energy/virial configuration occupy `N_i + 2` lines. The first line should have 7 numbers. The first one is the total energy of the current configuration. The remaining 6 numbers are the xx, yy, zz, xy, yz, and zx components of the virial tensor of the current configuration. The second line gives the cell vectors, similar to the first line for the force configurations. The remaining `N_i` lines are similar to those in the force configurations, but each line only has 4 numbers, which are the atom type and positions components.

3.3 Output files of GPUGA

For each simulation, four output files will be generated in the working directory: `ga.out`, `force.out`, `energy.out`, and `virial.out`. Repeating a simulation will remove the existing results and only keep the new results. A Matlab script `plot_results.m` can be used to plot a figure similar to Fig. 2 in Fan et al. (2019).

3.3.1 The ga.out file

This file will contain `maximum_generation` lines (or less if the simulation is terminated before completion) and 11 columns. Each line contains the following items

```
step best_fitness D0 alpha r0 S n beta h R1 R2
```

Here, `step` is the step in the GA evolution, starting with 0, `best_fitness` is the fitness function (the smaller, the better) for the elite (the best solution) in each generation, and the remaining 9 numbers are the potential parameters (in the same order as in `potential.in`) for the elite in each generation.

To find the best solution for one simulation, just check the last line.

3.3.2 The force.out file

There are 6 columns. The first three columns are the x, y, and z force components in the force configurations calculated from the best solution. The last three columns are the corresponding forces from `train.in`. The first `N_1` rows correspond to the `N_1` atoms in the first force configuration; the next `N_2` rows correspond to the `N_2` atoms in the second force configuration; and so on. Remember that the number of force configurations `Nc_force` is specified in `train.in`.

3.3.3 The energy.out file

There are 2 columns. The first column gives the energies calculated from the best solution. The second column gives the corresponding energies from `train.in`. Each row corresponds to one energy/virial configuration in `train.in`. Remember that the number of energy/virial configurations is `Nc - Nc_force`, which should be the number of rows in `energy.out`.

3.3.4 The virial.out file

There are 2 columns. The first column gives the virials calculated from the best solution. The second column gives the corresponding virials from `train.in`. The number of rows

is `(Nc - Nc_force) x 6`. The first 1/6 corresponds to the xx component of the viral in the same order as the energy data in `energy.out`, followed by the yy, zz, xy, yz, and zx components.

Bibliography

Zheyong Fan, Yanzhou Wang, Xiaokun Gu, Ping Qian, Yanjing Su, and Tapio Ala-Nissila. A minimal Tersoff potential for diamond silicon with improved descriptions of elastic and phonon transport properties. *arXiv preprint arXiv:1909.11474*, 2019. URL https://arxiv.org/abs/1909.11474.