GPUMD:
Graphics Processing Units
Molecular Dynamics

Reference Manual

Version 1.0

(July 12, 2017)

Authors:
Zheyong Fan, Ville Vierimaa, Mikko Ervasti, Ari Harju
Department of Applied Physics, Aalto University

Contents

Chapter 1

Introduction

1.1 What is GPUMD?

GPUMD stands for **G**raphics **P**rocessing **U**nits **M**olecular **D**ynamics. It is a new molecular dynamics (MD) code implemented fully on graphics processing units (GPUs). It was firstly used for heat transport simulations only but we are now making it more and more general.

1.2 Citations

If you use GPUMD in your published work, we kindly ask you to cite the following paper which describes the central algorithms used in GPUMD:

- Zheyong Fan, Wei Chen, Ville Vierimaa, and Ari Harju. Efficient molecular dynamics simulations with many-body potentials on graphics processing units. *Computer Physics Communications*, 218:10-16, 2017.

If your work involves using heat current and virial stress formulas as implemented in GPUMD, the following paper can be cited:

- Zheyong Fan, Luiz Felipe C. Pereira, Hui-Qiong Wang, Jin-Cheng Zheng, Davide Donadio, and Ari Harju. Force and heat current formulas for many-body potentials in molecular dynamics simulations with applications to thermal conductivity calculations. Phys. Rev. B, 92:094301, Sep 2015.

You can cite the following paper if you use GPUMD to study heat transport using the in-out decomposition for 2D materials and/or the spectral decomposition method as described in it:

- Zheyong Fan, Luiz Felipe C. Pereira, Petri Hirvonen, Mikko M. Ervasti, Ken R. Elder, Davide Donadio, Tapio Ala-Nissila, and Ari Harju. Thermal conductivity decomposition in two-dimensional materials: Application to graphene. Phys. Rev. B, 95:144309, Apr 2017.

1.3 Feedbacks

You can e-mail the first author 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 addresses can be used:

- zheyong.fan(at)aalto.fi (should be valid up to the end of 2018)
- brucenju(at)gmail.com
- zheyongfan(at)163.com

1.4 Acknowledgments

We acknowledge the computational resources provided by Aalto Science-IT project and Finland's IT Center for Science (CSC). We also thank the great help from the CUDA experts from NVIDIA and CSC during the GPU hackathon (13-09-2016 to 16-09-2016) organized by Sebastian von Alfthan.

Chapter 2

Features of GPUMD

2.1 GPU-accelerated force evaluation for many-body potentials

One of the major features of GPUMD is that force evaluation for many-body potentials has been significantly accelerated by using GPUs. Our efficient and flexible GPU-implementation of the force evaluation for many-body potentials relies on a set of simple expressions for force, virial stress, and heat current derived in Ref. [8]. Detailed algorithms for efficient CUDA-implementation have been presented in Ref. [6].

Using the methods as described in Refs. [8, 6], we have implemented various many-body potentials in GPUMD, including:

- The EAM-type potential with some analytical forms [31, 3].
- The Tersoff (1989) potential with single or double atom types [27].
- The Stillinger-Weber (1985) potential [25].

More many-body potentials will be implemented in GPUMD in future versions.

2.2 Utilities for heat transport simulations

Apart from being highly efficient, another unique feature of GPUMD is that it has useful utilities to study heat transport. The current version of GPUMD can calculate the following quantities related to heat transport:

- It can calculate the phonon density of states (DOS) from the velocity autocorrelation function (VAC), using the method of Dickey and Paskin [5].
- It can calculate equilibrium heat current auto-correlation (HAC), whose time integral gives the running thermal conductivity according to the Green-Kubo relation [11, 16]. As stressed in Ref. [8], the heat current as implemented in LAMMPS [22] does not apply to many-body potentials and significantly underestimates the thermal conductivity in 2D materials described by many-body potentials. GPUMD also contains the thermal conductivity decomposition method as introduced in Ref. [7], which is essential for 2D materials.

- It can calculate the thermal conductivity of a system of finite length or the thermal boundary resistance (Kapitza resistance) of an interface or similar structures using non-equilibrium MD (NEMD) methods. The spectral decompositions method as described in Ref. [7] has also been implemented.

2.3 Other features

2.3.1 Boundary conditions

GPUMD supports the following boundary conditions in each direction:

- free boundary conditions
- periodic boundary conditions (using the minimum image convention)
- fixed boundary conditions (by fixing some atoms)

The current version of GPUMD only supports orthogonal simulation boxes. Triclinic simulation boxes *might* be implemented in a future version.

2.3.2 Neighbor list construction

GPUMD has the following two versions for neighbor list construction and automatically chooses an appropriate one according to the inputs:

- an $O(N^2)$ method which builds the Verlet neighbor list by directly checking the distance between one particle and all the other particles in the simulation box
- an $O(N)$ method which first builds a cell list and then converts to the Verlet neighbor list

When the neighbor list is required to be updated during a simulation, one only has to specify a skin distance and the code will automatically determine when the neighbor list needs to be updated.

2.3.3 Integration methods

The velocity-Verlet [26] integration scheme is used for all the ensembles. The supported ensembles and the adopted methods are

- the NVE ensemble
- the NVT ensemble
  - the Berendsen method [1]
  - the Nosé-Hoover chain method [21, 13, 19, 18, 29] with a fixed chain length of 4.
- the NPT ensemble
  - the Berendsen method [1] with the pressures in each direction controlled independently

We are working on implementing more integration methods.

2.4 Major changes compared to the previous version

This is the very first version.

Chapter 3

Theoretical formalisms and numerical algorithms

3.1 Physical units used in the program

The basic units in the numerical calculations are chosen to be

1. Energy: eV (electron volt)
2. Length: Å (angstrom)
3. Mass: amu (atomic mass unit)
4. Temperature: K (kelvin)
5. Charge: e (elementary charge)

The purpose of using these units is to make the values of most quantities in the code close to unity. The units for all the other quantities are thus fixed. Here are some examples:

1. Time: Å $\text{amu}^{1/2}$ $\text{eV}^{-1/2}$, which is about 1.018051×10^1 fs
2. Velocity: $\text{eV}^{1/2}$ $\text{amu}^{-1/2}$
3. Force: eV Å^{-1}
4. Pressure (stress): eV Å^{-3}, which is about 1.602177×10^2 GPa
5. Thermal conductivity: $\text{eV}^{3/2}$ $\text{amu}^{-1/2}$ Å^2 K^{-1} which is about 1.573769×10^5 W m^{-1} K^{-1}
6. Boltzmann's constant: $k_B \approx 8.617343 \times 10^{-5}$ eV K^{-1}
7. Electrostatic constant: $k_C = \frac{1}{4\pi\epsilon_0} \approx 1.441959 \times 10^1$ eV Å e^{-2}

Important note: The input and output files do not necessarily adopt these units. For example, time step in the input file is in unit of fs, rather than Å $\text{amu}^{1/2}$ $\text{eV}^{-1/2}$. Details on the units adopted by the input and output files are presented in Chapter 5.

3.2 Overall structure of the program

GPUMD is written using CUDA C/C++. The overall style is C, without using classes (except for some C-style structures), but we might switch to C++ style in the future. Except for data initialization and some calculations that are very cheap or inherently serial, all the other calculations are done on the GPU.

The current version of GPUMD has 53 source files. Except for the `main.cu` file, all the other files come in pairs (e.g., `gpumd` below means `gpumd.cu` and `gpumd.h`). The logical dependence of the files can be summarized as follows:

- `main` is the entrance of the program and depends on `gpumd`
- `gpumd` depends on
  - `initialize`
  - `finalize`
  - `run`
- `run` depends on
  - `potential`
  - `velocity`
  - `parse`
  - `dump`
  - `hac`
  - `shc`
  - `vac`
  - `heat`
  - `integrate`
  - `neighbor`
  - `validate`
  - `force`
- `integrate` depends on `force`
- `validate` depends on `force`
- `neighbor` depends on
  - `neighbor_ON1`
  - `neighbor_ON2`
- `force` depends on
  - `lj1`
  - `ri`
  - `eam_zhou_2004`

  - `eam_dai_2006`
  - `sw_1985`
  - `tersoff_1989_1`
  - `tersoff_1989_2`

- All the files (except for `main`) depend on `common`

3.3 Neighbour list construction

We use the Verlet neighbor list when evaluating the forces between particles. Two methods for constructing the Verlet neighbor list are implemented, one is an $O(N^2)$ method and the other is an $O(N)$ method.

3.3.1 A simple quadratic-scaling method

In the $O(N^2)$ method, the Verlet neighbor list is constructed by directly checking the distance between every pair of particles. Therefore, the computational effort scales as N^2, where N is the number of particles. This method only requires a single CUDA kernel. Algorithm 1 presents a pseudo code for the CUDA kernel.

Algorithm 1 The $O(N^2)$ method of neighbour list construction

Require: b is the block index
Require: t is the thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$ is the particle index
Require: N is the number of particles
Require: r_c^2 is the square of the cutoff distance for building the neighbor list
Require: NN_i is the number neighbors for particle i
Require: NL_{ik} is the index of the kth neighbor of particle i
Require: $\boldsymbol{r}_i$ is the position vector of particle i

```
1:  k ← 0
2:  if i < N then
3:      load r_i from the global memory
4:      for j = 0 to N − 1 do
5:          if j = i then
6:              continue
7:          end if
8:          load r_j from the global memory and calculate r_ij = r_j − r_i
9:          apply the minimum image convention to r_ij
10:         if |r_ij|^2 < r_c^2 then
11:             NL_ik ← j
12:             k ← k + 1
13:         end if
14:     end for
15:     NN_i ← k
16: end if
```

In this kernel, the block size is S_b and the grid size is $\lceil N/S_b \rceil$. The **if** statement is used to avoid manipulating invalid memory.

3.3.2 A linear-scaling method

In this method, one partitions the system into cells and only searches for neighbors of a given particle in a small number of cells. In 3D, the there are $3^3 = 27$ cells to be searched, which does not scale with N. The overall computational effort of this method scales as $27N_0N \sim N$, where N_0 is the average number of particles in one cell. Therefore, this is a linear-scaling, or $O(N)$ method.

When periodic boundary conditions are applied, the number of cells in each direction is determined as

$$N_x = \lfloor L_x/r_c \rfloor; \tag{3.1}$$

$$N_y = \lfloor L_y/r_c \rfloor; \tag{3.2}$$

$$N_z = \lfloor L_z/r_c \rfloor. \tag{3.3}$$

Here, L_x, L_y, and L_z are the box lengths. If a direction has free boundary conditions, we set the number of cells in that direction to 1. The total number of cells is thus $N_c = N_xN_yN_z$.

With the number of cells determined, we next determine the number of particles C_n ($n = 0, 1, \cdots, N_c - 1$) in each cell. We need to use a CUDA kernel to do this. A pseudo code for the kernel is presented in Algorithm 2.

Algorithm 2 Determine the number of particles in each cell

Require: b is the block index
Require: t is the thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$ is the particle index
Require: N is the number of particles
Require: C_n is the number of particles in cell n and has been initialized to 0
1: **if** $i < N$ **then**
2: calculate cell index n of particle i
3: atomic operation: $C_n \leftarrow C_n + 1$
4: **end if**

Note that atomic operations (for integer data) are used to avoid write conflict. In the above kernel, we need to calculate the cell index of a given particle (with coordinate components x, y, and z). This is done by using a device function. The total cell index n is calculated from three indices:

$$n_x = \lfloor x/r_c \rfloor; \tag{3.4}$$

$$n_y = \lfloor y/r_c \rfloor; \tag{3.5}$$

$$n_z = \lfloor z/r_c \rfloor; \tag{3.6}$$

$$n = n_x + N_x n_y + N_x N_y n_z. \tag{3.7}$$

The cell index in the x-direction is required to be no less then 0 and no larger than $N_x - 1$. That is, when $n_x < 0$, we increase n_x by N_x; when $n_x \geq N_x$, we decrease n_x by N_x. The other directions have similar requirements.

We then calculate the prefix sum (exclusive scan) S_n of C_n:

$$S_0 = 0; \tag{3.8}$$

$$S_n = \sum_{m=0}^{n-1} C_n \quad (0 \le n \le N_c). \tag{3.9}$$

For this, we use the `thrust::exclusive_scan` function from the thrust library.

Now we can determine which particles are in which cells. We define a one-dimensional array I of length N, with I_{S_n} to $I_{S_n+C_n-1}$ being the indices of the particles in cell n. This array is constructed by using a CUDA kernel and a corresponding pseudo code is presented in Algorithm 3.

Algorithm 3 Determine the array I containing the particle indices in the order of increasing cell index

Require: b is the block index
Require: t is the thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$ is the particle index
Require: N is the number of particles
Require: C_n is the number of particles in cell n and has been initialized to 0
Require: S_n is the prefix sum of C_n as defined in the text
Require: I_{S_n} to $I_{S_n+C_n-1}$ are indices of the particles in cell n
1: **if** $i < N$ **then**
2: calculate cell index n of particle i
3: $I_{S_n+C_n} \leftarrow i$
4: atomic operation: $C_n \leftarrow C_n + 1$
5: **end if**

Up to now, the so-called cell list has been constructed. The remaining task is to convert the cell list to the Verlet neighbor list. This can be done by a CUDA kernel similar to that for the $O(N^2)$ method. A pseudo code is presented in Algorithm 4.

We note that the computation time used for the construction of the cell list is negligible compared to that used for the construction of the Verlet neighbor list from the cell list. However, we have to do this conversion because our efficient force evaluation algorithm [6] requires using the Verlet neighbor list rather than the cell list. Fortunately, the neighbor list usually only needs to be updated every tens of time steps with a typical skin distance (defined as the difference between the cutoff distance used for building the neighbor list and the cutoff distance used for force evaluation). In some simulations such as calculating the thermal conductivity of stable solids, the neighbor list even does not need to be updated during the simulation.

3.3.3 How to choose between the two versions?

The $O(N)$ version is faster than the $O(N^2)$ version in most cases. But sometimes we still use the $O(N^2)$ version. Here are the choices made in GPUMD:

- If the number of cells in any direction with periodic boundary conditions is less than 3, the $O(N)$ version is not applicable (as some neighbors will be counted twice) and

Algorithm 4 Construct the Verlet neighbour list from the cell list

Require: b is the block index
Require: t is the thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$ is the particle index
Require: N is the number of particles
Require: r_c^2 is the square of the cutoff distance for building the neighbor list
Require: C_n is the number of particles in cell n and has been initialized to 0
Require: S_n is the prefix sum of C_n as defined in the text
Require: I_{S_n} to $I_{S_n+C_n-1}$ are the indices of the particle in cell n
Require: NN_i is the number neighbors for particle i
Require: NL_{ik} is the index of the kth neighbor of particle i
Require: $\boldsymbol{r}_i$ is the position vector of particle i

```
k ← 0
if i < N then
    load r_i from the global memory
    calculate cell index n of particle i
    for m in all the neighbor cells of cell n (including cell n) do
        for l = 0 to C_m − 1 do
            j ← I_{S_m+l}
            if j = i then
                continue
            end if
            load r_j from the global memory and calculate r_ij = r_j − r_i
            apply the minimum image convention to r_ij
            if |r_ij|^2 < r_c^2 then
                NL_ik ← j
                k ← k + 1
            end if
        end for
        NN_i ← k
    end for
end if
```

the $O(N^2)$ version will be used. Therefore, if you hope the $O(N)$ version will be used, you should make sure that the number of cells in any direction with periodic boundary conditions is no less than 3.

- The $O(N^2)$ version is only faster when the number of cells is very small. Take a 3D system with periodic boundary conditions in each direction for example, when the number of cells in each direction is 3, the $O(N^2)$ version is definitely faster than the $O(N)$ version, but the $O(N)$ version is already faster when the number of cells in each direction is 4. After doing some tests, we have decided to use the $O(N)$ version whenever the total number of cells is larger than 50. This might not be always optimal but is not a bad choice.

- The $O(N^2)$ version is deterministic, but the $O(N)$ version contains randomness, due to the use of atomic operations in the CUDA kernels. Using atomic operations, the order of the neighbor particles for a given particle can be different from run to

run, which is not desirable for the purpose of debugging. In view of this, we have provided a compiling option (see Chapter 5 for details) to switch on the debugging mode, where the $O(N^2)$ version is always used.

In summary, GPUMD chooses an appropriate method for neighbor list construction automatically and no input is expected from the users.

3.3.4 How often should the neighbor list be updated?

The frequency of updating the neighbor list depends on the applications. If one simulates a stable solid system with the initial neighbor list containing all the neighbors that has possible interactions with a given particle during the whole simulation, the neighbor list does not need to be updated at all. In other cases, the neighbor list needs to be updated during the simulation. Usually, one can set an updating frequency such as 10, which means that the neighbor list will be updated every 10 integration steps. However, this can be either inefficient or unsafe. Another way is to determine at every integration step whether the neighbor list needs to be updated by checking how far each atom has moved since the last neighbor list updating. It can be argued that the neighbor list should be updated when the maximum traveling distance of the particles since the last updating exceeds half of the skin distance (set by the user). As this check takes negligible time, GPUMD uses this method to determine automatically when the neighbor list is to be updated and thus does not expect the users to specify an updating frequency.

3.4 General formalisms for force evaluation and related calculations

3.4.1 General form of empirical potential functions

In classical molecular dynamics, the total potential energy U of a system can be written as the sum of site potentials U_i:

$$U = \sum_{i=1}^{N} U_i. \tag{3.10}$$

The site potential can have different forms in different potential models. Although there are numerous potential models proposed to date, they can be largely classified into two groups: two-body potentials and many-body potentials.

3.4.2 Force

For two-body potentials, the site potential U_i can be expressed as

$$U_i = \frac{1}{2} \sum_{j \neq i} U_{ij}(r_{ij}). \tag{3.11}$$

Here, $r_{ij} = |\boldsymbol{r}_j - \boldsymbol{r}_i|$ is the distance between particles i and j and $U_{ij}(r_{ij})$ is the pair potential between them. The total force acted on particle i can be derived to be:

$$\boldsymbol{F}_i = -\nabla_i U = \sum_{j \neq i} \frac{\partial U_{ij}(r_{ij})}{\partial r_{ij}} \frac{\boldsymbol{r}_{ij}}{r_{ij}}. \tag{3.12}$$

In this manual, we use the symbol $\boldsymbol{r}_{ij}$ to denote the position difference vector from particle i to particle j:

$$\boxed{\boldsymbol{r}_{ij} \equiv \boldsymbol{r}_j - \boldsymbol{r}_i}. \tag{3.13}$$

The reader should bear this in mind when comparing the formulas in this manual with those in the literature, because many authors have used the opposite sign convention. One can also write the total force on particle i in the following form:

$$\boldsymbol{F}_i = \sum_{j \neq i} \boldsymbol{F}_{ij}, \tag{3.14}$$

where

$$\boldsymbol{F}_{ij} = \frac{\partial U_{ij}(r_{ij})}{\partial r_{ij}} \frac{\boldsymbol{r}_{ij}}{r_{ij}} \tag{3.15}$$

is the pairwise force acting on particle i by particle j. Newton's third law is apparently valid here, in the sense that

$$\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji}. \tag{3.16}$$

In some many-body potentials such as the embedded-atom method potential [4], the site potential can not be written in the form of Eq. (3.11). In some other many-body potentials such as the Tersoff potential, the site potential can be written in the form of Eq. (3.11), but the U_{ij} in this equation does not only depend on the distance between particles i and j. The force formulas for many-body potentials have confused the community a lot. Recently, a well-defined force expression for general many-body potentials that explicitly respects Newton's third law has been derived as [8]:

$$\boldsymbol{F}_i = \sum_{j \neq i} \boldsymbol{F}_{ij}, \tag{3.17}$$

where

$$\boxed{\boldsymbol{F}_{ij} = -\boldsymbol{F}_{ji} = \frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} - \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}} = \frac{\partial \left(U_i + U_j\right)}{\partial \boldsymbol{r}_{ij}}}. \tag{3.18}$$

Here, $\partial U_i / \partial \boldsymbol{r}_{ij}$ is a shorthand notation for a vector with cartesian components $\partial U_i / \partial x_{ij}$, $\partial U_i / \partial y_{ij}$, and $\partial U_i / \partial z_{ij}$. This pairwise force expression for many-body potentials has been confirm by Hardy [12] as well as by Chen and Diaz [2]. We have also confirmed its correctness by comparing with finite-difference calculations. This simple pairwise force expression for many-body potentials is the key for deriving well-defined expressions for other useful quantities such as virial stress and heat current, as discussed below.

3.4.3 Stress

Stress (tensor) is an important quantity in MD simulations. It consists of two parts: a virial part which is related to the force and an ideal-gas part which is related to the temperature. The virial part must be calculated along with force evaluation.

The validity of Newton's third law is crucial in simplifying the calculation of the virial stress. We know that the virial stress tensor is defined as

$$\mathbf{W} = \sum_i \mathbf{W}_i, \tag{3.19}$$

$$\mathbf{W}_i = \boldsymbol{r}_i \otimes \boldsymbol{F}_i. \tag{3.20}$$

Here, $\mathbf{W}_i$ can be regarded as the per-atom virial stress. For periodic systems, the presence of absolute positions $\boldsymbol{r}_i$ would cause problems. However, when Newton's third law is valid, one can rewrite the per-atom virial stress as

$$\boxed{\mathbf{W}_i = -\frac{1}{2}\sum_{j\neq i} \boldsymbol{r}_{ij} \otimes \boldsymbol{F}_{ij}}, \tag{3.21}$$

where only relative positions $\boldsymbol{r}_{ij}$ are involved. Because Newton's third law also applies to many-body potentials, the above expression of virial stress is valid for any classical potential.

The ideal-gas part of the stress is isotropic, which is given by the ideal-gas pressure:

$$p_{\text{ideal}} = \frac{Nk_BT}{V}, \tag{3.22}$$

where N is the number of particles, k_B is Boltzmann's constant, T is the absolute temperature, and V is the volume of the system.

Combining the ideal-gas part and the virial part, the total stress tensor $\sigma^{\alpha\beta}$ can be expressed as:

$$\sigma^{\alpha\beta} = -\frac{1}{2V}\sum_i \sum_{j\neq i} r_{ij}^{\alpha} F_{ij}^{\beta} + \frac{Nk_BT}{V}\delta^{\alpha\beta}. \tag{3.23}$$

Here, α and β can be x, y, and z and $\delta^{\alpha\beta}$ is the Kronecker symbol. We will denote the diagonal part of the total stress tensor as a "vector" $\boldsymbol{p}$ with components $p_x = \sigma^{xx}$, $p_y = \sigma^{yy}$, and $p_z = \sigma^{zz}$. If the system is isotropic, we usually average the diagonal terms to get a scalar:

$$p = \frac{1}{3}\left(p_x + p_y + p_z\right) = -\frac{1}{6V}\sum_i \sum_{j\neq i} \boldsymbol{r}_{ij} \cdot \boldsymbol{F}_{ij} + \frac{Nk_BT}{V}. \tag{3.24}$$

3.4.4 Heat current

GPUMD can be used to compute the lattice thermal conductivity using the Green-Kubo [11, 16] formula, which requires calculating the heat current.

In classical physics, the total heat current vector $\boldsymbol{J}$ of a system is defined to be[1] the time derivative of the sum of the energy moments:

$$\boldsymbol{J} = \frac{d}{dt}\sum_i \boldsymbol{r}_i E_i. \tag{3.25}$$

Here, E_i is the site energy of particle i, which is the sum of the kinetic and potential energies:

$$E_i = \frac{1}{2} m_i \boldsymbol{v}_i^2 + U_i. \tag{3.26}$$

Using Lebniz's rule, we have

$$\boldsymbol{J} = \sum_i \boldsymbol{v}_i E_i + \sum_i \boldsymbol{r}_i \frac{d}{dt} E_i. \tag{3.27}$$

[1]Actually, it is J/V that has the dimension of heat current density (also called heat flux), which has the unit of W m^{-2} in the international unit system. However, it is tedious to add the factor of $1/V$ in many of the subsequent equations.

The first term on the right hand side is usually called the convective term and we do not need to evaluate it in the force-evaluation kernel. The second term,

$$\boldsymbol{J}^{\text{pot}} = \sum \boldsymbol{r}_i \frac{dE_i}{dt}, \tag{3.28}$$

is usually called the potential term and needs to be evaluated in the force-evaluation kernel.

When using the Green-Kubo method, we need to use periodic boundary conditions (at least in the transport directions). For two-body potentials, we can arrive at the following expression which is suitable for implementation:

$$\boldsymbol{J}^{\text{pot}} = -\frac{1}{2} \sum_i \sum_{j \neq i} \boldsymbol{r}_{ij} \left(\boldsymbol{F}_{ij} \cdot \boldsymbol{v}_i \right). \tag{3.29}$$

This equation can be expressed in an equivalent way:

$$\boldsymbol{J}^{\text{pot}} = -\frac{1}{2} \sum_i \sum_{j \neq i} \left(\boldsymbol{r}_{ij} \otimes \boldsymbol{F}_{ij} \right) \cdot \boldsymbol{v}_i. \tag{3.30}$$

Therefore, we can also write it in terms of the per-atom virial:

$$\boldsymbol{J}^{\text{pot}} = \sum_i \mathbf{W}_i \cdot \boldsymbol{v}_i. \tag{3.31}$$

We can also define the per-atom heat current $\boldsymbol{J}_i^{\text{pot}}$ for the potential part in the following way:

$$\boldsymbol{J}^{\text{pot}} = \sum_i \boldsymbol{J}_i^{\text{pot}}; \tag{3.32}$$

$$\boldsymbol{J}_i^{\text{pot}} = \mathbf{W}_i \cdot \boldsymbol{v}_i. \tag{3.33}$$

However, we note that the above formula only applies to two-body potentials. For many-body potentials, it has been demonstrated [8] that the above virial-based formula is wrong and the correct one is

$$\boxed{\boldsymbol{J}_i^{\text{pot}} = \sum_{j \neq i} \boldsymbol{r}_{ij} \left(\frac{\partial U_j}{\partial \boldsymbol{r}_{ji}} \cdot \boldsymbol{v}_i \right)}. \tag{3.34}$$

The above heat current formula is usually applied in equilibrium simulations. In nonequilibrium simulations, the following expression for the nonequilibrium heat flux (heat current density) [7] from a subsystem A to a subsystem B is more useful:

$$\boxed{Q_{A \to B} = -\frac{1}{S} \sum_{i \in A} \sum_{j \in B} \left\langle \left(\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} \cdot \boldsymbol{v}_j - \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}} \cdot \boldsymbol{v}_i \right) \right\rangle}, \tag{3.35}$$

where S is the cross-sectional area perpendicular to the direction of the heat flow. This formula applies to general many-body potentials. For two-body potentials, it reduces to the following one:

$$Q_{A \to B}^{\text{two-body}} = -\frac{1}{2S} \sum_{i \in A} \sum_{j \in B} \left\langle \boldsymbol{F}_{ij} \cdot \left(\boldsymbol{v}_i + \boldsymbol{v}_j \right) \right\rangle. \tag{3.36}$$

3.5 Integration by one step

The aim of time evolution is to find the phase trajectory

$$\{\boldsymbol{r}_i(t_1),\ \boldsymbol{v}_i(t_1)\}_{i=1}^N,\ \{\boldsymbol{r}_i(t_2),\ \boldsymbol{v}_i(t_2)\}_{i=1}^N,\ \cdots \tag{3.37}$$

starting from the initial phase point

$$\{\boldsymbol{r}_i(t_0),\ \boldsymbol{v}_i(t_0)\}_{i=1}^N. \tag{3.38}$$

The time interval between two time points $\Delta t = t_1 - t_0 = t_2 - t_1 = \cdots$ is called the time step.

The algorithm for integrating by one step depends on the ensemble type and other external conditions. We discuss them in detail below. There are many ensembles used in MD simulations, but we only consider the following 3 in the current version:

- The NVE ensemble, where the particle number N, the system volume V, and the total energy E are kept constant. It is also called the micro-canonical ensemble.
- The NVT ensemble, where the particle number N, the system volume V, and the temperature T are kept constant. It is also called the canonical ensemble.
- The NPT ensemble, where the particle number N, the pressure p, and the temperature T are kept constant. There seems to be no simple name for this important ensemble, but it is usually called the isothermal-isobaric ensemble.

3.5.1 The NVE ensemble and the velocity-Verlet algorithm

In the NVE ensemble, the dynamics of the system is Hamiltonian and the equations of motion can be derived from Hamilton's equations. Because these equations of motion have the time-reversal symmetry, a good numerical integrating method (an integrator) should preserve this symmetry.

One of the most widely used integrators which has the property of time-reversibility is the so-called velocity-Verlet method [26]. This integrator is also symplectic. These two properties make the velocity-Verlet integrator very stable for long-time simulations. Here are the velocity and position updating equations in the velocity-Verlet method:

$$\boldsymbol{v}_i(t_{m+1}) \approx \boldsymbol{v}_i(t_m) + \frac{\boldsymbol{F}_i(t_m) + \boldsymbol{F}_i(t_{m+1})}{2m_i}\Delta t; \tag{3.39}$$

$$\boldsymbol{r}_i(t_{m+1}) \approx \boldsymbol{r}_i(t_m) + \boldsymbol{v}_i(t_m)\Delta t + \frac{1}{2}\frac{\boldsymbol{F}_i(t_m)}{m_i}(\Delta t)^2, \tag{3.40}$$

where m_i is the mass of particle i.

The above velocity-Verlet integrator can be derived by finite-difference method (Taylor series expansion), but a more general method, which can be generalized to more sophisticated situations, is the classical time-evolution operator approach, or the Liouville operator approach [29]. In this approach, the time-evolution of a classical system by one step can be formally expressed as

$$\begin{pmatrix} \boldsymbol{r}_i(t+\Delta t) \\ \boldsymbol{p}_i(t+\Delta t) \end{pmatrix} = e^{iL\Delta t} \begin{pmatrix} \boldsymbol{r}_i(t) \\ \boldsymbol{p}_i(t) \end{pmatrix}, \tag{3.41}$$

where $\boldsymbol{p}_i$ is the momentum of particle i and $e^{iL\Delta t}$ is called the classical evolution operator, which is the classical counterpart of the quantum evolution operator. The operator iL in the exponent of the evolution operator is called the Liouville operator and is defined by

$$iL(\text{anything}) = \{\text{anything}, H\} \equiv \sum_{i=1}^{N} \left(\frac{\partial H}{\partial \boldsymbol{p}_i} \cdot \frac{\partial}{\partial \boldsymbol{r}_i} - \frac{\partial H}{\partial \boldsymbol{r}_i} \cdot \frac{\partial}{\partial \boldsymbol{p}_i} \right) (\text{anything}). \tag{3.42}$$

Here, H is the Hamiltonian of the system. Because

$$\frac{\partial H}{\partial \boldsymbol{p}_i} = \frac{\boldsymbol{p}_i}{m_i} \text{ and } -\frac{\partial H}{\partial \boldsymbol{r}_i} = \boldsymbol{F}_i, \tag{3.43}$$

we have

$$iL = iL_1 + iL_2, \tag{3.44}$$

$$iL_1 = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i}{m_i} \cdot \frac{\partial}{\partial \boldsymbol{r}_i}, \tag{3.45}$$

$$iL_2 = \sum_{i=1}^{N} \boldsymbol{F}_i \cdot \frac{\partial}{\partial \boldsymbol{p}_i}. \tag{3.46}$$

Here, we have divided the Liouville operator into two parts. In general, iL_1 and iL_2 do not commute, and therefore $e^{iL\Delta t} \neq e^{iL_1\Delta t} e^{iL_2\Delta t}$. However, there is an important theorem called the Trotter theorem, which can be used to derive the following approximation:

$$e^{iL\Delta t} \approx e^{iL_2\Delta t/2} e^{iL_1\Delta t} e^{iL_2\Delta t/2}. \tag{3.47}$$

Now, we can express the one-step integration as

$$\begin{pmatrix} \boldsymbol{r}_i(t+\Delta t) \\ \boldsymbol{p}_i(t+\Delta t) \end{pmatrix} \approx e^{iL_2\Delta t/2} e^{iL_1\Delta t} e^{iL_2\Delta t/2} \begin{pmatrix} \boldsymbol{r}_i(t) \\ \boldsymbol{p}_i(t) \end{pmatrix}. \tag{3.48}$$

To make further derivations, we note that for an arbitrary constant c, we have

$$e^{c\frac{\partial}{\partial x}} x = x + c. \tag{3.49}$$

Applying this identity to the right most operator in the above equation, we have

$$\begin{pmatrix} \boldsymbol{r}_i(t+\Delta t) \\ \boldsymbol{p}_i(t+\Delta t) \end{pmatrix} \approx e^{iL_2\Delta t/2} e^{iL_1\Delta t} \begin{pmatrix} \boldsymbol{r}_i(t) \\ \boldsymbol{p}_i(t) + \frac{\Delta t}{2} \boldsymbol{F}_i(t) \end{pmatrix}. \tag{3.50}$$

Then, applying the operator $e^{iL_1\Delta t}$, we have

$$\begin{pmatrix} \boldsymbol{r}_i(t+\Delta t) \\ \boldsymbol{p}_i(t+\Delta t) \end{pmatrix} \approx e^{iL_2\Delta t/2} \begin{pmatrix} \boldsymbol{r}_i(t) + \Delta t \frac{\boldsymbol{p}_i(t) + \frac{\Delta t}{2} \boldsymbol{F}_i(t)}{m_i} \\ \boldsymbol{p}_i(t) + \frac{\Delta t}{2} \boldsymbol{F}_i(t) \end{pmatrix}. \tag{3.51}$$

Last, applying the remaining operator $e^{iL_2\Delta t/2}$, we have

$$\begin{pmatrix} \boldsymbol{r}_i(t+\Delta t) \\ \boldsymbol{p}_i(t+\Delta t) \end{pmatrix} \approx \begin{pmatrix} \boldsymbol{r}_i(t) + \Delta t \frac{\boldsymbol{p}_i(t) + \frac{\Delta t}{2} \boldsymbol{F}_i(t)}{m_i} \\ \boldsymbol{p}_i(t) + \frac{\Delta t}{2} \boldsymbol{F}_i(t) + \frac{\Delta t}{2} \boldsymbol{F}_i(t+\Delta t) \end{pmatrix}. \tag{3.52}$$

It is clear that this equation is equivalent to Eqs. (3.39) and (3.40).

We can see that in the velocity-Verlet integrator, the position updating can be done in one step, but the velocity updating can only be done by two steps, one before force updating and the other after it. Algorithm 5 gives the pseudo code for the complete time-stepping in the NVE ensemble, including force updating.

Algorithm 5 The whole time-stepping in the NVE ensemble.

1: update the velocities partially

$$\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \frac{1}{2}\frac{\boldsymbol{F}_i}{m_i}\Delta t \tag{3.53}$$

2: update the positions completely

$$\boldsymbol{r}_i \leftarrow \boldsymbol{r}_i + \boldsymbol{v}_i \Delta t \tag{3.54}$$

3: update the forces

$$\boldsymbol{F}_i \leftarrow \boldsymbol{F}_i(\{\boldsymbol{r}_i\}) \tag{3.55}$$

4: complete updating the velocities

$$\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \frac{1}{2}\frac{\boldsymbol{F}_i}{m_i}\Delta t \tag{3.56}$$

3.5.2 Berendsen thermostat and barostat

Using the Berendsen thermostat, the integration algorithm in the NVT ensemble only requires an extra scaling of all the velocity components, as shown in Algorithm 6. For the NPT ensemble, the Berendsen barostat requires an extra scaling of positions and box lengths, as shown in Algorithm 7. The Berendsen thermostat and barosat are very suitable for equilibrating the system to a target temperature and pressure.

Algorithm 6 The whole time-stepping in the NVT ensemble using the Berendsen method.

1: perform the whole time-stepping for the NVE ensemble as shown in Algorithm 5
2: scale the velocities

Algorithm 7 The whole time-stepping in the NPT ensemble using the Berendsen method.

1: perform the whole time-stepping for the NVE ensemble as shown in Algorithm 5
2: scale the velocities
3: scale the positions and box lengths

The velocities are scaled in the Berendsen thermostat in the following way:

$$\boldsymbol{v}_i^{\text{scaled}} = \boldsymbol{v}_i \sqrt{1 + \alpha_T \left(\frac{T_0}{T} - 1\right)}. \tag{3.57}$$

Here, α_T is a dimensionless parameter, T_0 is the target temperature, and T is the instant temperature calculated from the current velocities $\{\boldsymbol{v}_i\}$. The parameter α_T should be positive and not larger than 1. When $\alpha_T = 1$, the above formula reduces to the simple velocity-scaling formula:

$$\boldsymbol{v}_i^{\text{scaled}} = \boldsymbol{v}_i \sqrt{\frac{T_0}{T}}. \tag{3.58}$$

A smaller α_T represents a weaker coupling between the system and the thermostat. Practically, any value of α_T in the range of $0.001 \sim 1$ can be used.

In the Berendsen barostat algorithm, the particle positions and box length in a given direction are scaled if periodic boundary conditions are applied to that direction. The scaling of the positions reads

$$\boldsymbol{r}_i^{\text{scaled}} = \boldsymbol{r}_i \left[1 - \alpha_p(\boldsymbol{p}_0 - \boldsymbol{p})\right]. \tag{3.59}$$

Here, α_p is a parameter and $\boldsymbol{p}_0$ ($\boldsymbol{p}$) is the target (instant) pressure in the three directions. The parameter α_p is not dimensionless, and it requires some try-and-error to find a good value of it for a given system. A harder/softer system requires a smaller/larger value of α_p. In the unit system adopted by GPUMD, it is recommended that $\alpha_p = 10^{-4} \sim 10^{-2}$. Only directions with periodic boundary conditions will be affected by the barostat.

3.5.3 Nosé-Hoover chain thermostat

The Nosé-Hoover chain method [21, 13, 19, 18, 29] is more suitable for calculating equilibrium properties in a specific ensemble. In the current version of GPUMD, only the Nosé-Hoover chain thermostat is implemented. We hope to implement the Nosé-Hoover chain barostat in a future version.

The equations of motion in the Nosé-Hoover chain method are

$$\frac{d}{dt}\boldsymbol{r}_i = \frac{\boldsymbol{p}_i}{m_i}, \tag{3.60}$$

$$\frac{d}{dt}\boldsymbol{p}_i = \boldsymbol{F}_i - \frac{\pi_0}{Q_0}\boldsymbol{p}_i, \tag{3.61}$$

$$\frac{d}{dt}\eta_k = \frac{\pi_k}{Q_k} \ (k = 0, 1, \cdots, M-1), \tag{3.62}$$

$$\frac{d}{dt}\pi_0 = 2\left(\sum_i \frac{\boldsymbol{p}_i^2}{2m_i} - dN\frac{k_B T}{2}\right) - \frac{\pi_1}{Q_1}\pi_0, \tag{3.63}$$

$$\frac{d}{dt}\pi_k = 2\left(\frac{\pi_{k-1}^2}{2Q_{k-1}} - \frac{k_B T}{2}\right) - \frac{\pi_{k+1}}{Q_{k+1}}\pi_k \ (k = 1, 2, \cdots, M-2), \tag{3.64}$$

$$\frac{d}{dt}\pi_{M-1} = 2\left(\frac{\pi_{M-2}^2}{2Q_{M-2}} - \frac{k_B T}{2}\right). \tag{3.65}$$

The optimal choice [19] for the thermostat masses is

$$Q_0 = dNk_B T\tau^2, \tag{3.66}$$

$$Q_k = k_B T\tau^2 \ (k = 1, 2, \cdots, M-1), \tag{3.67}$$

where τ is a time parameter, whose value is usually chosen by try and error in practice. A good choice is $\tau = 100\Delta t$, where Δt is the time step for integration.

An integration scheme for the NVT ensemble using the Nosé-Hoover chain can also be formulated using the approach of the time-evolution operator [18, 29]. The total Liouville operator for the equations of motion in the Nosé-Hoover chain method is [18, 29]

$$iL = iL_1 + iL_2 + iL_T, \tag{3.68}$$

$$iL_1 = \sum_{i=1}^{N} \frac{\boldsymbol{p}_i}{m_i} \cdot \frac{\partial}{\partial \boldsymbol{r}_i}, \tag{3.69}$$

$$iL_2 = \sum_{i=1}^{N} \boldsymbol{F}_i \cdot \frac{\partial}{\partial \boldsymbol{p}_i}. \tag{3.70}$$

$$iL_T = \sum_{k=0}^{M-1} \frac{\pi_k}{Q_k} \frac{\partial}{\partial \eta_k} + \sum_{k=0}^{M-2} \left(G_k - \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \right) \frac{\partial}{\partial \pi_k} + G_{M-1} \frac{\partial}{\partial \pi_{M-1}} - \sum_{i=0}^{N-1} \frac{\pi_0}{Q_0} \boldsymbol{p}_i \cdot \frac{\partial}{\partial \boldsymbol{p}_i}. \tag{3.71}$$

That is, the Liouville operator for the NVT ensemble contains an extra term iL_T related to the thermostat variables, which is absent from that for the NVE ensemble.

The total time-evolution operator $e^{iL\Delta t}$ for one step can be factorized using the Trotter theorem as in the case of the NVE ensemble:

$$e^{iL} \approx e^{iL_T \Delta t/2} e^{iL_2 \Delta t/2} e^{iL_1 \Delta t} e^{iL_2 \Delta t/2} e^{iL_T \Delta t/2}. \tag{3.72}$$

Comparing this with the factorization in the NVE ensemble, we see that we only need to apply the operator $e^{iL_T \Delta t/2}$ before and after applying the usual velocity-Verlet integrator in the NVE ensemble.

The operator $e^{iL_T \Delta t/2}$ can be further factorized into some elementary factors using the Trotter theorem. First, we define the following decomposition of the operator iL_T:

$$iL_T = iL_{T1} + iL_{T2} + iL_{T3}, \tag{3.73}$$

$$iL_{T1} = \sum_{k=0}^{M-1} \frac{\pi_k}{Q_k} \frac{\partial}{\partial \eta_k}, \tag{3.74}$$

$$iL_{T2} = \sum_{k=0}^{M-2} \left(G_k - \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \right) \frac{\partial}{\partial \pi_k} + G_{M-1} \frac{\partial}{\partial \pi_{M-1}}, \tag{3.75}$$

$$iL_{T3} = -\sum_{i=0}^{N-1} \frac{\pi_0}{Q_0} \boldsymbol{p}_i \cdot \frac{\partial}{\partial \boldsymbol{p}_i}. \tag{3.76}$$

We can then make the following factorization:

$$e^{iL_T \Delta t/2} \approx e^{iL_{T2} \Delta t/4} e^{iL_{T3} \Delta t/2} e^{iL_{T1} \Delta t/2} e^{iL_{T2} \Delta t/4}. \tag{3.77}$$

There are still a few terms in iL_{T2} and we need to factorize $e^{iL_{T2} \Delta t/4}$ further. We can factorize the $e^{iL_{T2} \Delta t/4}$ term on the right of the above equation as

$$e^{iL_{T2} \Delta t/4} \approx \prod_{k=0}^{M-2} \left(e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} e^{\frac{\Delta t}{4} G_k \frac{\partial}{\partial \pi_k}} e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} \right) e^{\frac{\Delta t}{4} G_{M-1} \frac{\partial}{\partial \pi_{M-1}}} \tag{3.78}$$

and correspondingly factorize that on the left as

$$e^{iL_{T2} \Delta t/4} \approx e^{\frac{\Delta t}{4} G_{M-1} \frac{\partial}{\partial \pi_{M-1}}} \prod_{k=M-2}^{0} \left(e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} e^{\frac{\Delta t}{4} G_k \frac{\partial}{\partial \pi_k}} e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} \right). \tag{3.79}$$

It can be shown that the effect of the operator $e^{cx\frac{\partial}{\partial x}}$ on x is to scale it by a factor of e^c:

$$e^{cx\frac{\partial}{\partial x}} x = e^c x. \tag{3.80}$$

Algorithm 8 The whole time-stepping in the NVT ensemble using the Nosé-Hoover chain method.

1: apply the operator $e^{iL_T\Delta t/2}$ except for $e^{iL_{T3}\Delta t/2}$ within it and save the value of $e^{-(\pi_0/Q_0)\Delta t/2}$
2: scale the velocity components of all the particles by the factor $e^{-(\pi_0/Q_0)\Delta t/2}$
3: perform the whole time-stepping for the NVE ensemble as shown in Algorithm 5
4: apply the operator $e^{iL_T\Delta t/2}$ except for $e^{iL_{T3}\Delta t/2}$ within it and save the value of $e^{-(\pi_0/Q_0)\Delta t/2}$
5: scale the velocity components of all the particles by the factor $e^{-(\pi_0/Q_0)\Delta t/2}$

Therefore, the effect of the operator $e^{iL_{T3}\Delta t/2}$ is to scale the momenta of all the particles in the system by a uniform factor $e^{-(\pi_0/Q_0)\Delta t/2}$. Although this operator appears in the factorization of $e^{iL_T\Delta t/2}$, its does not affect the thermostat variables. Therefore, when applying the operator $e^{iL_T\Delta t/2}$, we only need to update the variables related to the thermostats and save this factor for later use when we update the variables for the particles. In this way, the update for the thermostat variables and that for the particle variables are separated. Algorithm 8 presents the pseudo code for the whole time-stepping in the NVT ensemble using the Nosé-Hoover chain method.

There are also some other tricks in the algorithm. For details, we refer to the excellent book by Tuckerman [29].

3.6 Heat transport

3.6.1 Heat current autocorrelation and lattice thermal conductivity

In MD simulations, a popular approach of computing the lattice thermal conductivity is to use the Green-Kubo formula [11, 16]. In this method, the running lattice thermal conductivity along the x-direction (similar expressions apply to other directions) can be expressed as an integral of the heat current autocorrelation (HAC):

$$\boxed{\kappa_{xx}(t) = \frac{1}{k_B T^2 V} \int_0^t dt' \mathrm{HAC}_{xx}(t')}. \tag{3.81}$$

Here, k_B is Boltzmann's constant, V is the volume of the simulated system, T is the absolute temperature, and t is the correlation time. The HAC is

$$\boxed{\mathrm{HAC}_{xx}(t) = \langle J_x(0) J_x(t) \rangle}, \tag{3.82}$$

where $J_x(0)$ and $J_x(t)$ are the total heat current of the system at two time points separated by an interval of t. The symbol $\langle\rangle$ means that the quantity inside will be averaged over different time origins.

The calculation of the heat current $\boldsymbol{J}$ has been discussed earlier. Here, we assume that we have calculated the total heat current of the system at M number of time points and saved them into the global memory. The time interval $\Delta\tau$ between the time points here needs not to be the same as the time step Δt used in the time-stepping. Usually, $\Delta\tau = 10\Delta t$ is a good choice. From the N_d heat current data, we can calculate at most N_d HAC data $\mathrm{HAC}_{xx}(t)$, with $t = 0, \Delta\tau, 2\Delta\tau, \cdots, (N_d - 1)\Delta\tau$. However, a correlation

function becomes more and more noisy as the correlation time increases and in practical applications, one has to make sure that the production time $N_d\Delta\tau$ is much larger than the maximum correlation time $t_{\rm max}$ one needs. The number of HAC data N_c is related to the maximum correlation time by $t_{\rm max} = N_c\Delta\tau$. In most cases, $N_c = N_d/10$ is a good choice. It is also convenient to use the same number of time origins, $N_d - N_c$, to do the time-average for each correlation time. With these considerations, we arrive at the following explicit expression for the HAC:

$$\mathrm{HAC}_{xx}(n_c\Delta\tau) = \frac{1}{N_d - N_c}\sum_{m=0}^{N_d-N_c-1} J_x(m\Delta\tau)J_x((m+n_c)\Delta\tau), \tag{3.83}$$

where $n_c = 0, 1, 2, \cdots, N_c - 1$.

Because the HAC at different correlation times can be calculated independently, we can simply use one CUDA-block for one point of the HAC data. Shared memory is used to do the summation in an binary-reduction way. An algorithm for the CUDA implementation can be found in Ref. [9].

Last, we note that for 2D materials, the heat current can be naturally decomposed into an in-plane component and an out-of-plane component. The lattice thermal conductivity can be decomposed accordingly. See Ref. [7] for more details.

3.6.2 NEMD method

The thermal conductivity of a finite-length system can also be computed by the NEMD method. In this method, a temperature gradient is established by externally generating a non-equilibrium heat current. If the steady-state heat flux is Q and the established temperature gradient is ∇T, the thermal conductivity is calculated according to Fourier's law as

$$\boxed{\kappa = \frac{Q}{|\nabla T|}}. \tag{3.84}$$

The heat flux can be generated in various ways, such as the velocity re-scaling method [14, 15] and the momentum swapping method [20]. However, some guesswork is needed in choosing appropriate parameters in these methods. Another method is to coupled the source/sink region to a thermostat with a higher/lower temperature. By setting the temperature difference, one has better control to the temperature gradient. When steady state is achieved, a temperature gradient will be established and the heat flux Q can be calculated from the energy transfer rate dE/dt between the source/sink and the thermostat:

$$\boxed{Q = \frac{dE/dt}{S}}, \tag{3.85}$$

where S is the cross-sectional area perpendicular to the transport direction. It has been shown [7] that Eq. (3.85) gives consistent heat flux as calculated by Eq. (3.35). As in the case of the Green-Kubo method, the nonequilibrium heat flux defined in Eq. (3.35) can also be decomposed into in-pane and out-of-plane components for 2D materials. See Ref. [7] for details.

We have tested that all these NEMD methods can give consistent results. To make the code simpler, we have only kept the version using thermostats in GPUMD.

In the framework of the NEMD method, one can also calculate spectrally decomposed thermal conductivity (or conductance) using the method as described in Ref. [7]. This

method is base on the works by Sääskilahti *et al.* [23, 24] and Zhou *et al.* [32] but the formulation in Ref. [7] is simpler and does not use the harmonic approximation. In this method, one first calculates the following correlation function:

$$\boxed{K_{A\to B}(t) = -\frac{1}{S}\sum_{i\in A}\sum_{j\in B}\left\langle\left(\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}}(0)\cdot\boldsymbol{v}_j(t) - \frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}(t)\cdot\boldsymbol{v}_i(t)\right)\right\rangle}, \tag{3.86}$$

which reduces to the nonequilibrium heat flux as defined in Eq. (3.35). Then one can define the following Fourier transform pairs:

$$\boxed{\tilde{K}_{A\to B}(\omega) = \int_{-\infty}^{\infty} dt e^{i\omega t} K_{A\to B}(t)}; \tag{3.87}$$

$$\boxed{K_{A\to B}(t) = \int_{0}^{\infty} \frac{d\omega}{2\pi} e^{-i\omega t} \tilde{K}_{A\to B}(\omega)}. \tag{3.88}$$

By setting $t = 0$ in the equation above, we can get the following spectral decomposition of the nonequilibrium heat flux:

$$\boxed{Q_{A\to B} = \int_{0}^{\infty} \frac{d\omega}{2\pi}\left[2\tilde{K}_{A\to B}(\omega)\right]}. \tag{3.89}$$

From the spectral decomposition of the nonequilibrium heat flux, one can deduce the spectrally decomposed thermal conductivity corresponding to a given finite system with a temperature gradient ∇T,

$$\boxed{\kappa(\omega) = \frac{2\tilde{K}_{A\to B}(\omega)}{|\nabla T|} \quad \text{with} \quad \kappa = \int_{0}^{\infty} \frac{d\omega}{2\pi}\kappa(\omega)}, \tag{3.90}$$

or a spectrally decomposed thermal conductance corresponding to a given segment or junction (interface) with a temperature difference ΔT,

$$\boxed{G(\omega) = \frac{2\tilde{K}_{A\to B}(\omega)}{|\Delta T|} \quad \text{with} \quad G = \int_{0}^{\infty} \frac{d\omega}{2\pi}G(\omega)}. \tag{3.91}$$

For 2D materials, one can further consider the in-out decomposition. See Ref. [7] for details.

3.7 Velocity autocorrelation and related quantities

Velocity autocorrelation (VAC) is an important quantity in MD simulations. On the one hand, its integral with respect to the correlation time gives the running diffusion constant, which is equivalent to that obtained by a time derivative of the mean square displacement (MSD). On the other hand, its Fourier transform is the phonon density of states (PDOS) [5].

3.7.1 Running diffusion coefficient

The VAC is a single-particle correlation function. This means that we can define the VAC for individual particles. For particle i, the VAC along the x direction is defined as

$$\langle v_{xi}(0)v_{xi}(t)\rangle. \tag{3.92}$$

Then, one can define the mean VAC for any number of particles. In the current version of GPUMD, it is assumed that one wants to calculate the mean VAC in the whole simulated system:

$$\boxed{\mathrm{VAC}_{xx}(t) = \frac{1}{N}\sum_{i=1}^{N}\langle v_{xi}(0)v_{xi}(t)\rangle}. \tag{3.93}$$

The order between the time-average (denoted by $\langle\rangle$) and the space-average (the average over the particles) can be changed:

$$\boxed{\mathrm{VAC}_{xx}(t) = \left\langle \frac{1}{N}\sum_{i=1}^{N} v_{xi}(0)v_{xi}(t) \right\rangle}. \tag{3.94}$$

Using the same conventions as in the case of HAC calculations, we have the following explicit expression for the VAC:

$$\mathrm{VAC}_{xx}(n_c\Delta\tau) = \frac{1}{(N_d - N_c)N}\sum_{m=0}^{N_d-N_c-1}\sum_{i=1}^{N} v_{xi}(m\Delta\tau)v_{xi}((m+n_c)\Delta\tau), \tag{3.95}$$

where $n_c = 0, 1, 2, \cdots, N_c - 1$. The algorithm for calculating the VAC is quite similar to that for calculating the HAC and it thus omitted.

After obtaining the VAC, we can calculate the running diffusion constant $D_{xx}(t)$ as

$$\boxed{D_{xx}(t) = \int_0^t dt'\ \mathrm{VAC}_{xx}(t')}. \tag{3.96}$$

One can prove that this is equivalent to the time-derivative of the MSD, i.e., the Einstein formula:

$$\boxed{D_{xx}(t) = \frac{1}{2}\frac{d}{dt}\Delta x^2(t)}, \tag{3.97}$$

where the MSD $\Delta x^2(t)$ is defined as

$$\boxed{\Delta x^2(t) = \left\langle \frac{1}{N}\sum_{i=1}^{N}\left[x_i(t) - x_i(0)\right]^2 \right\rangle = \frac{1}{N}\sum_{i=1}^{N}\left\langle \left[x_i(t) - x_i(0)\right]^2 \right\rangle}. \tag{3.98}$$

Here is the proof. Starting from the relation between position and velocity,

$$x_i(t) - x_i(0) = \int_0^t dt' v_{xi}(t'), \tag{3.99}$$

we have

$$\left[x_i(t) - x_i(0)\right]^2 = \int_0^t dt' v_{xi}(t') \int_0^t dt'' v_{xi}(t'') = \int_0^t dt' \int_0^t dt'' v_{xi}(t')v_{xi}(t''). \tag{3.100}$$

Then, the MSD can be expressed as

$$\Delta x^2(t) = \frac{1}{N}\sum_{i=1}^{N}\int_0^t dt' \int_0^t dt'' \left\langle v_{xi}(t')v_{xi}(t'')\right\rangle . \tag{3.101}$$

Using Lebniz's rule, we have

$$D_{xx}(t) = \frac{1}{2}\frac{d}{dt}\Delta x^2(t) = \frac{1}{N}\sum_{i=1}^{N}\int_0^t dt' \left\langle v_{xi}(t)v_{xi}(t')\right\rangle , \tag{3.102}$$

which can be rewritten as

$$D_{xx}(t) = \frac{1}{N}\sum_{i=1}^{N}\int_0^t dt' \left\langle v_{xi}(0)v_{xi}(t'-t)\right\rangle . \tag{3.103}$$

Letting $\tau = t' - t$, we get (note that here t is considered as a constant)

$$D_{xx}(t) = \frac{1}{N}\sum_{i=1}^{N}\int_{-t}^0 d\tau \left\langle v_{xi}(0)v_{xi}(\tau)\right\rangle , \tag{3.104}$$

which can be rewritten as

$$D_{xx}(t) = \frac{1}{N}\sum_{i=1}^{N}\int_{-t}^0 d\tau \left\langle v_{xi}(-\tau)v_{xi}(0)\right\rangle . \tag{3.105}$$

Letting $t' = -\tau$, we finally get

$$D_{xx}(t) = \frac{1}{N}\sum_{i=1}^{N}\int_0^t dt' \left\langle v_{xi}(t')v_{xi}(0)\right\rangle = \int_0^t dt' \ \mathrm{VAC}_{xx}(t'). \tag{3.106}$$

We thus have derived the Green-Kubo formula from the Einstein formula.

In summary,

- The derivative of half of the MSD gives the running diffusion coefficient.
- The integral of the VAC gives the running diffusion coefficient.
- One can obtain the MSD by integrating the VAC twice (numerically).

3.7.2 Phonon density of states

It is interesting that the same VAC can be used to compute the PDOS, as first demonstrated by Dickey and Paskin [5]. The PDOS is simply the Fourier transform of the normalized VAC:

$$\rho_x(\omega) = \int_{-\infty}^{\infty} dt e^{i\omega t} \ \mathrm{VAC}_{xx}(t). \tag{3.107}$$

Here, $\mathrm{VAC}_{xx}(t)$ should be understood as the normalized function $\mathrm{VAC}_{xx}(t)/\mathrm{VAC}_{xx}(0)$. Although it looks simple, it does not mean that one can get the correct PDOS by a naive fast Fourier transform (FFT) routine. Actually, this computation is very cheap and we

do not need FFT at all. What we need is a discrete cosine transform. To see this, we first note that, by definition, $\mathrm{VAC}_{xx}(-t) = \mathrm{VAC}_{xx}(t)$. Using this, we have

$$\rho_x(\omega) = \int_{-\infty}^{\infty} dt \cos(\omega t)\ \mathrm{VAC}_{xx}(t). \tag{3.108}$$

Because we only have the VAC data at the N_c discrete time points, the above integral is approximated by the following discrete cosine transform:

$$\rho_x(\omega) \approx \sum_{n_c=0}^{N_c-1} (2-\delta_{n_c0}) \Delta\tau \cos(\omega n_c \Delta\tau)\ \mathrm{VAC}_{xx}(n_c\Delta\tau). \tag{3.109}$$

Here, δ_{n_c0} is the Kronecker δ function and the factor $(2-\delta_{n_c0})$ accounts for the fact that there is only one point for $t=0$ and there are two equivalent points for $t \neq 0$. Last, we note that a window function is needed to suppress the unwanted Gibbs oscillation in the calculated PDOS. In GPUMD, the Hann window $H(n_c)$ is applied:

$$\rho_x(\omega) \approx \sum_{n_c=0}^{N_c-1} (2-\delta_{n_c0}) \Delta\tau \cos(\omega n_c \Delta\tau)\ \mathrm{VAC}_{xx}(n_c\Delta\tau) H(n_c); \tag{3.110}$$

$$H(n_c) = \frac{1}{2}\left[\cos\left(\frac{\pi n_c}{N_c}\right) + 1\right]. \tag{3.111}$$

Here are some comments on the normalization of the PDOS. In the literature, one usually uses an arbitrary unit for the PDOS, but it actually has a dimension of [time], and an appropriate unit for it can be 1/THz or ps. The normalization of $\rho_x(\omega)$ can be determined by the inverse Fourier transform:

$$\mathrm{VAC}_{xx}(t) = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi} e^{-i\omega t} \rho_x(\omega). \tag{3.112}$$

As we have normalized the VAC, we have

$$1 = \mathrm{VAC}_{xx}(0) = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi} \rho_x(\omega). \tag{3.113}$$

Because $\rho_x(-\omega) = \rho_x(\omega)$, we have

$$\int_{0}^{\infty} \frac{d\omega}{2\pi} \rho_x(\omega) = \frac{1}{2}. \tag{3.114}$$

The calculated PDOS should meet this normalization condition (approximately).

Chapter 4

Potential models implemented in GPUMD

In this chapter, we discuss in detail the potential models that have been implemented in GPUMD. The formats of the potential files for the potential models are also presented. In the potential files, the unit of energy is eV and the unit of length is Å. One should be able to deduce the dimension of all parameters from the relevant equations defining them.

4.1 Conventions

The following conventions are used in this chapter:

- The position difference vector from particle i to particle j is denoted

$$\boxed{\boldsymbol{r}_{ij} \equiv \boldsymbol{r}_j - \boldsymbol{r}_i}. \tag{4.1}$$

  The component form of this equation is

$$x_{ij} = x_j - x_i;\ y_{ij} = y_j - y_i;\ z_{ij} = z_j - z_i. \tag{4.2}$$

- The angle formed by $\boldsymbol{r}_{ij}$ and $\boldsymbol{r}_{ik}$ is denoted

$$\boxed{\cos\theta_{ijk} = \cos\theta_{ikj} = \frac{\boldsymbol{r}_{ij} \cdot \boldsymbol{r}_{ik}}{r_{ij} r_{ik}}}. \tag{4.3}$$

- the derivative $\partial/\partial \boldsymbol{r}_{ij}$ should be understood as a vector operator with components $\partial/\partial x_{ij}$, $\partial/\partial y_{ij}$, and $\partial/\partial z_{ij}$.

- It is easy to verify that

$$\boxed{\frac{\partial r_{ij}}{\partial \boldsymbol{r}_{ij}} = \frac{\boldsymbol{r}_{ij}}{r_{ij}}}. \tag{4.4}$$

- It is also easy to verify that

$$\boxed{\frac{\partial \cos\theta_{ijk}}{\partial \boldsymbol{r}_{ij}} = \frac{1}{r_{ij}}\left[\frac{\boldsymbol{r}_{ik}}{r_{ik}} - \frac{\boldsymbol{r}_{ij}}{r_{ij}}\cos\theta_{ijk}\right]}. \tag{4.5}$$

- If f is a function of x, then $f'(x)$ is understood as $\partial f/\partial x$.

In the force evaluation kernel for any potential model, the key quantities to be calculated are $\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}}$ and $\frac{\partial U_j}{\partial \boldsymbol{r}_{ji}}$, the latter being related to the former by an exchange of the indices, $i \leftrightarrow j$. We thus need to derive an explicit expression of $\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}}$ for each potential model. This is trivial for two-body potentials and we only present expressions for many-body potentials. We will call $\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}}$ the **partial force**.

4.2 The Lennard-Jones potential

The Lennard-Jones potential is one the most simplest two-body potentials used in MD simulations. The pair potential between particle i and j is

$$U_{ij} = 4\epsilon \left(\frac{\sigma^{12}}{r_{ij}^{12}} - \frac{\sigma^6}{r_{ij}^6} \right). \tag{4.6}$$

The current version of GPUMD only implements the single-element version of the Lennard-Jones potential with simple cutoff (without potential and/or force shifts). More variants will be added in the future.

The potential file for this potential model reads

```
lj1
epsilon sigma cutoff
```

Here, `cutoff` is the cutoff distance.

4.3 The rigid-ion potential

By rigid-ion potential, we mean a potential model consisting of a short range part in the Buckingham form

$$U_{ij} = A_{ij} \exp\left(-b_{ij} r_{ij}\right) - \frac{C_{ij}}{r_{ij}^6} \tag{4.7}$$

and a Coulomb potential. The Coulomb potential is evaluated using the damped-shifted-force (DSF) method by Fennell and Gezelter [10], which is based on the Wolf summation method [30]. The DSF version of the pairwise Coulomb potential can be written as:

$$U_{ij} = \frac{q_i q_j}{4\pi\epsilon_0} \left[\frac{\operatorname{erfc}(\alpha r_{ij})}{r_{ij}} - \frac{\operatorname{erfc}(\alpha R_c)}{R_c} + \left(\frac{\operatorname{erfc}(\alpha R_c)}{R_c^2} + \frac{2\alpha}{\sqrt{\pi}} \frac{\exp(-\alpha^2 R_c^2)}{R_c} \right) (r_{ij} - R_c) \right], \tag{4.8}$$

where α and R_c are the electrostatic damping factor and the cutoff radius, respectively. A good choice is $\alpha = 0.2$ and $R_c \geq 15$ Å. In GPUMD, we fix α to 0.2 and let the users to specify the cutoff distance.

The potential file for this potential model reads

```
ri
q_1   q_2  cutoff
A_11, b_11 C_11
A_22, b_22 C_22
A_12, b_12 C_12
```

4.4 The EAM potential

This is a simple many-body potential. The site potential energy is

$$U_i = \frac{1}{2}\sum_{j\neq i}\phi(r_{ij}) + F(\rho_i). \tag{4.9}$$

Here, the part with $\phi(r_{ij})$ is a pairwise potential and $F(\rho_i)$ is the embedding potential, which depends on the electron density ρ_i at site i. This density is contributed by the neighbors of i:

$$\rho_i = \sum_{j\neq i} f(r_{ij}). \tag{4.10}$$

The many-body part of the EAM potential comes from the embedding potential. The **partial force** in our formulation is

$$\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} = \frac{1}{2}\phi'(r_{ij})\frac{\partial r_{ij}}{\partial \boldsymbol{r}_{ij}} + F'(\rho_i)f'(r_{ij})\frac{\partial r_{ij}}{\partial \boldsymbol{r}_{ij}}. \tag{4.11}$$

All the three functions above, $\phi(r_{ij})$, $F(\rho_i)$, and $f(r_{ij})$ are usually given as splines, but in the current version of GPUMD, we have only implemented two analytical forms.

4.4.1 The analytical form by Zhou *et al.*

The pair potential between two atoms of the same type a is

$$\phi^{aa}(r) = \frac{A^a \exp[-\alpha(r/r_e^a - 1)]}{1 + (r/r_e^a - \kappa^a)^{20}} - \frac{B^a \exp[-\beta(r/r_e^a - 1)]}{1 + (r/r_e^a - \lambda^a)^{20}}. \tag{4.12}$$

The contribution of the electron density from an atom of type a is

$$f^a(r) = \frac{f_e^a \exp[-\beta(r/r_e^a - 1)]}{1 + (r/r_e^a - \lambda^a)^{20}}. \tag{4.13}$$

The pair potential between two atoms of different types a and b is then constructed as

$$\phi^{ab}(r) = \frac{1}{2}\left[\frac{f^b(r)}{f^a(r)}\phi^{aa}(r) + \frac{f^a(r)}{f^b(r)}\phi^{bb}(r)\right]. \tag{4.14}$$

The embedding energy function is piecewise:

$$F(\rho) = \sum_{i=0}^{3} F_{ni}\left(\frac{\rho}{\rho_n} - 1\right)^i, \quad (\rho < 0.85\rho_e) \tag{4.15}$$

$$F(\rho) = \sum_{i=0}^{3} F_i\left(\frac{\rho}{\rho_e} - 1\right)^i, \quad (0.85\rho_e \leq \rho < 1.15\rho_e) \tag{4.16}$$

$$F(\rho) = F_e\left[1 - \ln\left(\frac{\rho}{\rho_s}\right)^\eta\right]\left(\frac{\rho}{\rho_s}\right)^\eta, \quad (\rho \geq 1.15\rho_e) \tag{4.17}$$

For each element, there are 20 parameters, which are r_e, f_e, ρ_e, ρ_s, α, β, A, B, κ, λ, F_{n0}, F_{n1}, F_{n2}, F_{n3}, F_0, F_1, F_2, F_3, η, and F_e. Parameter values for 16 metals are tabulated in the paper by Zhou *et al.* [31]. Although we have presented the potential in a form applicable to systems with more than one atom type, the current version of GPUMD only supports a single atom type for this potential model. Extensions will be made in a future version.

The potential file for this potential model reads

```
eam_zhou_2004
r_e
f_e
rho_e
rho_s
alpha
beta
A
B
kappa
lambda
F_n0
F_n1
F_n2
F_n3
F_0
F_1
F_2
F_3
eta
F_e
cutoff
```

The last parameter `cutoff` is the cutoff distance which is not intrinsic to the model. The order of the parameters is the same as in Table III of Ref. [31].

4.4.2 The extended Finnis-Sinclair potential by Dai *et al.*

This is a very simple EAM-type potential which is an extension of the Finnis-Sinclair potential. The function for the pair potential is

$$\phi(r) = \begin{cases} (r-c)^2 \sum_{n=0}^{4} c_n r^n & r \leq c \\ 0 & r > c \end{cases} \tag{4.18}$$

The function for the density is

$$\phi(r) = \begin{cases} (r-d)^2 + B^2(r-d)^4 & r \leq d \\ 0 & r > d \end{cases} \tag{4.19}$$

The function for the embedding energy is

$$F(\rho) = -A\rho^{1/2}. \tag{4.20}$$

For each element, there are 9 parameters, which are A, d, c, c_0, c_1, c_2, c_3, c_4, and B. The original Finnis-Sinclair potential corresponds to the case where the parameters c_3, c_4, and B are all zero. Parameter values for 12 metals can be found from the paper by Dai *et al.* [3].

The potential file for this potential model reads

```
eam_dai_2006
A
d
c
c_0
c_1
c_2
c_3
c_4
B
```

4.5 The Stillinger-Weber potential

Here we consider the original Stillinger-Weber potential as proposed by Stillinger and Weber in 1985 [25]. The total potential energy consists of a two-body part and a three-body part. The site potential is

$$U_i = \frac{1}{2}V_2(r_{ij}) + \frac{1}{2}\sum_{j\neq i}\sum_{k\neq i,j} h_{ijk}. \tag{4.21}$$

where the two-body part is

$$V_2(r_{ij}) = A\epsilon\left[B\left(\frac{\sigma}{r_{ij}}\right)^4 - 1\right]\exp\left(\frac{1}{r_{ij}/\sigma - a}\right) \tag{4.22}$$

and the three-body part is

$$h_{ijk} = \lambda\exp\left[\frac{\gamma}{r_{ij}/\sigma - a} + \frac{\gamma}{r_{ik}/\sigma - a}\right](\cos\theta_{ijk} - \cos\theta_0)^2. \tag{4.23}$$

Here, A, B, ϵ, σ, a, λ, γ, and $\cos\theta_0$ are parameters. Here is an explicit expression of the **partial force** for the three-body part:

$$\begin{aligned}\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} = &\left[\lambda\frac{\gamma}{r_{ij}/\sigma_{ij} - a_{ij}} + \frac{\gamma}{r_{ik}/\sigma_{ik} - a_{ik}}\right](\cos\theta_{ijk} - \cos\theta_0) \\ &\times\left[2\frac{\partial\cos\theta_{ijk}}{\partial\boldsymbol{r}_{ij}} - \frac{\gamma}{\sigma_{ij}\left(\frac{r_{ij}}{\sigma_{ij}} - a_{ij}\right)^2}(\cos\theta_{ijk} - \cos\theta_0)\frac{\partial r_{ij}}{\partial\boldsymbol{r}_{ij}}\right].\end{aligned} \tag{4.24}$$

The potential file for this potential model reads

```
sw
epsilon lambda A B a gamma sigma cos_theta_0
```

4.6 The Tersoff potential

There are many variants of the Tersoff potential. In the current version of GPUMD, we only consider the form as described in Ref. [27] published in 1989.

The site potential can be written as

$$U_i = \frac{1}{2} \sum_{j \neq i} f_C(r_{ij}) \left[f_R(r_{ij}) - b_{ij} f_A(r_{ij}) \right]. \tag{4.25}$$

The function f_C is a cutoff function, which is 1 when $r_{ij} < R_{ij}$ and 0 when $r_{ij} > S_{ij}$ and takes the following form in the intermediate region:

$$f_C(r_{ij}) = \frac{1}{2} \left[1 + \cos \left(\pi \frac{r_{ij} - R_{ij}}{S_{ij} - R_{ij}} \right) \right]. \tag{4.26}$$

The repulsive function f_R and the attractive function f_A take the following forms:

$$f_R(r) = A_{ij} e^{-\lambda_{ij} r_{ij}}; \tag{4.27}$$

$$f_A(r) = B_{ij} e^{-\mu_{ij} r_{ij}}. \tag{4.28}$$

The bond-order is

$$b_{ij} = \chi_{ij} \left(1 + \beta_i^{n_i} \zeta_{ij}^{n_i} \right)^{-\frac{1}{2n_i}}, \tag{4.29}$$

where

$$\zeta_{ij} = \sum_{k \neq i,j} f_C(r_{ik}) g_{ijk}, \tag{4.30}$$

and

$$g_{ijk} = 1 + \frac{c_i^2}{d_i^2} - \frac{c_i^2}{d_i^2 + (h_i - \cos \theta_{ijk})^2}. \tag{4.31}$$

Here, A_{ij}, B_{ij}, λ_{ij}, μ_{ij}, β_i, n_i, c_i, d_i, h_i, R_{ij}, S_{ij}, and χ_{ij} are material-specific parameters. The parameters values can be found in Tersoff's original paper [27] and a later erratum [28]. Note that when $i \neq j$, the following mixing rules are used to determining some parameter values:

$$A_{ij} = \sqrt{A_{ii} A_{jj}}; \tag{4.32}$$

$$B_{ij} = \sqrt{B_{ii} B_{jj}}; \tag{4.33}$$

$$R_{ij} = \sqrt{R_{ii} R_{jj}}; \tag{4.34}$$

$$S_{ij} = \sqrt{S_{ii} S_{jj}}; \tag{4.35}$$

$$\lambda_{ij} = (\lambda_{ii} + \lambda_{jj})/2; \tag{4.36}$$

$$\mu_{ij} = (\mu_{ii} + \mu_{jj})/2. \tag{4.37}$$

The parameter χ_{ij} is 1 for $i = j$ and can deviate slightly from 1 for $i \neq j$.

In the current version of GPUMD, one can simulate systems with two atom types (such as SiC and SiGe) as well as those with a single atom type.

An expression of the **partial force** has been presented in Ref. [8] and we repeat it here:

$$\begin{aligned}\frac{\partial U_i}{\partial \boldsymbol{r}_{ij}} &= \frac{1}{2} f_C'(r_{ij})[f_R(r_{ij}) - b_{ij} f_A(r_{ij})] \frac{\partial r_{ij}}{\partial \boldsymbol{r}_{ij}} \\ &+ \frac{1}{2} f_C(r_{ij})[f_R'(r_{ij}) - b_{ij} f_A'(r_{ij})] \frac{\partial r_{ij}}{\partial \boldsymbol{r}_{ij}} \\ &- \frac{1}{2} f_C'(r_{ij}) \sum_{k \neq i,j} f_C(r_{ik}) f_A(r_{ik}) b_{ik}' g_{ijk} \frac{\partial r_{ij}}{\partial \boldsymbol{r}_{ij}} \\ &- \frac{1}{2} f_C(r_{ij}) f_A(r_{ij}) b_{ij}' \sum_{k \neq i,j} f_C(r_{ik}) g_{ijk}' \frac{\partial \cos\theta_{ijk}}{\partial \boldsymbol{r}_{ij}} \\ &- \frac{1}{2} f_C(r_{ij}) \sum_{k \neq i,j} f_C(r_{ik}) f_A(r_{ik}) b_{ik}' g_{ijk}' \frac{\partial \cos\theta_{ijk}}{\partial \boldsymbol{r}_{ij}}. \end{aligned} \tag{4.38}$$

The potential file for the single-element version reads

```
tersoff_1989_1
A B lambda mu beta n c d h R S
```

The potential file for the double-element version reads

```
tersoff_1989_2
A_1 B_1 lambda_1 mu_1 beta_1 n_1 c_1 d_1 h_1 R_1 S_1
A_2 B_2 lambda_2 mu_2 beta_2 n_2 c_2 d_2 h_2 R_2 S_2
chi
```

Chapter 5

Using GPUMD

After downloading and unpacking GPUMD, one can see the following folders:

- `src`, which contains all the source files of GPUMD and a `makefile`
- `potentials`, which contains all the potential files we have prepared
- `examples`, which contains all the examples (each in a sub-folder) we have prepared
- `doc`, which contains the source files for the manual

5.1 Compile and run GPUMD

5.1.1 Compile GPUMD

To compile GPUMD, one just needs to go to the `src` directory and type `make`. one may want to first do `make clean`. When the compilation finishes, an excitable named `gpumd` will be generated in the `src` directory.

In the `makefle`, the default compiling flag is

```
CFLAGS = -O3 -use_fast_math -DUSE_DP -arch=sm_35
```

If you want to obtain a version using single-precision arithmetics for all the floating point calculations in the code, you can remove `-DUSE_DP` in the first line. If you want to obtain a version without any randomness in the calculations, which is suitable for debugging, you can add `-DDEBUG` in this line. The last option `-arch=sm_35` specifies the compute capability of the target GPU card (e.g., 3.5 for Tesla K40) you will use. Note that GPUs with compute capability less than 2.0 are not supported by GPUMD.

5.1.2 Run simulations with GPUMD

To run a simulation or a set of simulations with GPUMD, one needs to first prepare some input files. For any individual simulation, one needs to prepare a file named `xyz.in` and a file named `run.in`, and put them into the same directory. Then, one just needs an extra input file, which we call a "driver input file", to specify the path(s) of the folder(s) containing the input files. This "driver input file" should have the following format:

```
number_of_simulations
path_1
path_2
...
```

Let us consider two explicit examples. Consider a "driver input file" which reads

```
1
examples/lattice_constant/si_tersoff
```

This means that there will be one simulation and the input files (`xyz.in` and `run.in`) are prepared in the folder `examples/lattice_constant/si_tersoff`. One can also run multiple simulations using a single "driver input file". An example is:

```
2
examples/lattice_constant/si_tersoff
examples/lattice_constant/si_sw
```

In this case, it means that two sets of inputs will be processed consecutively.

Now we are ready to run the code. Suppose that the "driver input file" is named as `input` and is in the folder where we can see the `src` folder, we can run the code using the following command:

```
src/gpumd < input
```

Output files will be created in the folders containing the corresponding input files.

The next two sections are devoted to describing the structures of the input and output files, respectively.

5.2 Input files of GPUMD

5.2.1 The xyz.in input file

A file named `xyz.in` should be prepared and should have the following format (empty lines and comments are not allowed):

```
N      M       cutoff
pbc_x  pbc_y   pbc_z  L_x L_y L_z
type_1 group_1 mass_1 x_1 y_1 z_1
type_2 group_2 mass_2 x_2 y_2 z_2
...
type_N group_N mass_N x_N y_N z_N
```

We explain line by line:

- In the first line, `N` is the number of atoms, `M` is the maximum possible number of neighbor atoms for one atom, and `cutoff` is the initial cutoff distance used for building the neighbor list.

- In the second line, `pbc_x`, `pbc_y`, and `pbc_z` can only be 1 or 0. If `pbc_x` is 1, it means that periodic boundary conditions will be applied to the x direction; if `pbc_x` is 0, it means that free boundary conditions will be applied to the x direction. Similar descriptions apply to the other two directions. The next three items in the second line, `L_x`, `L_y`, and `L_z`, are the initial lengths of the (rectangular) simulation box along the x, y, and z directions, respectively.

- In the third line, `type_1`, `group_1`, and `mass_1` are respectively the type, group label, and mass of the first atom. The next three items, `x_1`, `y_1`, and `z_1`, are the coordinates of this atom.

- Similarly, the $(m+2)$th line gives the information for the mth atom. This file should have `N+2` lines.

The atom type will be used to determine which potential parameters to use. The group label will be used for some other purposes such as calculating the block temperatures in NEMD simulations or realize fixed boundaries. We use integers to record the atom types and group labels and the indices start from 0. Explicit examples will be presented in the next chapter.

The mass should be given in unit of the unified atomic mass unit (amu). The cutoff distance, box lengths and atom coordinates should be given in unit of angstrom (Å).

5.2.2 The run.in input file

Then, a file named `run.in` should also be prepared. In this input file, blank lines and lines starting with `#` are ignored. All the other lines should be of the following form:

```
keyword parameter_1 parameter_2 ...
```

The overall structure of a `run.in` file is as follows:

```
#------------------------------------------------------------------------
# First, write these two keywords in any order (group-1)
potential
velocity

# Then, write (all or part of) these keywords in any order (group-2)
ensemble
time_step
neighbor
fix
dump_thermo
dump_position
dump_velocity
dump_force
dump_potential
dump_virial
compute_temp
compute_shc
compute_vac
compute_hac
```

```
# Then write the keyword run (group-3)
run

# Now one can repeat the last two groups as many times as one wants
#-----------------------------------------------------------------
```

We now describe the use of the keywords in detail.

1. The `potential` keyword.

   This keyword only has one parameter, which is the file name (including the absolute or relative path) containing the information of the potential that the user wants to use. An example is

   ```
   potential potentials/tersoff/si_tersoff_1989_1.txt
   ```

   By writing this, one has to make sure that the file `si_tersoff_1989_1.txt` has been prepared in the folder `potentials/tersoff/`. The potential files are discussed in the previous chapter.

2. The `velocity` keyword.

   This keyword only has one parameter, which is the initial temperature of the system. For example, the command

   ```
   velocity 10
   ```

   means that one wants to set the initial temperature to 10 K.

3. The `ensemble` keyword.

   This keyword specifies the ensemble type (including external conditions such as local heating and cooling) and the relevant parameters. The number of parameters depends on the first parameter, which can be:

   - `nve`. This corresponds to the NVE ensemble and there is no need to further specify any other parameters. Therefore, the full command is

     ```
     ensemble nve
     ```

   - `nvt_ber`. This corresponds to the NVT ensemble using the Berendsen method. In this case, one needs to specify an initial target temperature `T_1`, a final target temperature `T_2`, and a parameter `T_coup` which reflects the strength of the coupling between the system and the thermostat. The full command is

     ```
     ensemble nvt_ber T_1 T_2 T_coup
     ```

     The target temperature (not the instant system temperature) will vary linearly from `T_1` to `T_2` during a run.

   - `nvt_nhc`. This corresponds to the NVT ensemble using the Nosé-Hoover chain method. The full command is

     ```
     ensemble nvt_nhc T_1 T_2 T_coup
     ```

- `npt_ber`. This corresponds to the NPT ensemble using the Berendsen method. In this case, apart from the same parameters as in the case of `nvt_ber`, one needs to further specify 3 target pressures, `Px`, `Py`, and `Pz`, and a pressure coupling constant `P_coup`. The full command is

  ```
  ensemble npt_ber T_1 T_2 T_coup Px Py Pz P_coup
  ```

- `heat_nhc`. This corresponds to heating a source region and simultaneously cooling a sink region using local Nosé-Hoover chain thermostats. The full command is

  ```
  ensemble heat_nhc T T_coup delta_T label_source label_sink
  ```

  The target temperatures in the source region (with label `label_source`) and sink region (with label `label_sink`) are `T + delta_T` and `T - delta_T`, respectively. Therefore, the temperature difference between the two regions is twice of `delta_T`.

The units of temperature and pressure for this keyword are K and GPa, respectively. The temperature coupling constant in the Berendsen method can be any positive number less than or equal to 1 and we recommend a value in the range of [0.01, 1]. A larger number results in a faster control of the temperature. The temperature coupling constant in the Nosé-Hoover chain method is in unit of the time step and is recommended to be in the range of [100, 1000]. Here, a larger number results in a slower control of the temperature. The pressure coupling constant in the Berenden method should be a small positive number in the unit system adopted by GPUMD. We recommend a value in the range of $[0.01, 0.0001]$. For a stiffer material (like diamond or graphene), one should use a smaller value. In practice, all these parameters should be determined by try and error.

4. The `time_step` keyword.

   This keyword only requires a single parameter, which is the time step for integration in unit of fs (10^{-15} s). For example, the command

   ```
   time_step 1
   ```

   means that the time step for the current run is 1 fs. Note that the value of time step does not need to be set for each run in a "run.in" file. If you do not set a new value of time step in a run, the value in the previous run will be used.

5. The `neighbor` keyword.

   This keyword only requires a single parameter, which is the skin distance (the difference between the cutoff distance used in neighbor list construction and that used in force evaluation) in unit of Å. For example, the command

   ```
   neighbor 1
   ```

   means that the neighbor list will be updated when necessary (the code determines automatically when the neighbor list needs to be updated) and the skin distance is 1 Å. If this keyword is absent in a run, the neighbor list will not be updated during the run.

6. The `dump_thermo`, `dump_position`, `dump_velocity`, `dump_force`, `dump_potential`, and `dump_virial` keywords.

   These keywords only requires a single parameter, which is the output frequency for the relevant quantities: global thermodynamic quantities for `dump_thermo`, per-atom positions for `dump_position`, per-atom velocities for `dump_velocity`, per-atom forces for `dump_force`, per-atom potential energies for `dump_potential`, and per-atom virial for `dump_virial`. For example, the command

   ```
   dump_thermo 1000
   ```

   means that the thermodynamic quantities will be written into the file `thermo.out` (in the folder which contains the `run.in` file) every 1000 steps. By default, GPUMD does not dump these quantities. For example, if there is no `dump_position` command for one run, positions will not be output for that run.

7. The `fix` keyword.

   This keyword requires a single parameter which is the label of the group in which the atoms are to be fixed (velocities and forces are set to zero such that the atoms in the group do not move). For example, command

   ```
   fix 0
   ```

   means that atoms in group `0` will be fixed during the current run.

8. The `compute_temp` keyword.

   This keyword is used to compute and output the block temperatures (local temperatures in each group) and requires a single parameter which is the sampling interval for the block temperatures. For example, the command

   ```
   compute_temp 1000
   ```

   means that the block temperatures will be computed every 1000 time steps.

9. The `compute_shc` keyword.

   This keyword is used to compute the nonequilibrium heat flux correlation function $K_{A\to B}(t)$ define in Eq. (3.86) and requires 5 parameters. The first parameter is the sampling interval between two correlation steps. The second parameter is the total correlation steps. The third parameter is the number of steps used to calculate one correlation function. The last two parameters are the labels of the groups A and B as used in Eq. (3.86). For example, the command

   ```
   compute_shc 1 500 5000 10 11
   ```

   means that (1) you want to do this calculation; (2) the relevant data will be sampled every step; (3) the maximum number of correlation steps is 500; (4) the correlation function $K_{A\to B}(t)$ will be calculated every 5000 steps (If the number of steps in this run is 100000, there will be $100000/5000 = 20$ independent correlation functions calculated); (5) the group label of A is 10 and that of B is 11. The results will be written into a file named `shc.out` in the same folder where you put your `run.in` file in.

10. The `compute_vac` keyword.

    This keyword is related to the calculations of VAC (velocity autocorrelation) and two other related quantities: RDC (running diffusion coefficient) and DOS (phonon density of states). If this keyword appears in a run, VAC and related quantities will be calculated in the run. This keyword requires 3 parameters. The first parameter for this keyword is the sampling interval of the velocity data. The second parameter is the maximum number of correlation steps. The third parameter is the maximum angular frequency $\omega_{\max} = 2\pi\nu_{\max}$ used in the DOS calculations. For example, the command

    ```
    compute_vac 5 200 350
    ```

    means that (1) you want to calculate the VAC and related quantities; (2) the velocity data will be recorded every 5 steps; (3) the maximum number of correlation steps is 200; (4) the maximum angular frequency you want to consider is $\omega_{\max} = 2\pi\nu_{\max} =$ 350 THz. The results will be written into a file named `vac.out` in the same folder where you put your `run.in` file in.

11. The `compute_hac` keyword.

    The `compute_hac` keyword is similar to the `compute_vac` keyword. It is used to calculate HAC (heat current autocorrelation) and RTC (running thermal conductivity). It has 3 parameters. The first parameter is the sampling interval for the heat current data. The second parameter is the maximum correlation steps. These two parameters are similar to those for the `compute_vac` keyword. The third parameter for `compute_hac` is the output interval of the HAC and RTC data. For example, the command

    ```
    compute_hac 20 50000 10
    ```

    means that (1) you want to calculate the thermal conductivity using the Green-Kubo method; (2) the heat current data will be recorded every 20 steps; (3) the maximum number of correlation steps is 50000; (4) the HAC/RTC data will be averaged for every 10 data and the number of HAC/RTC data output in a given direction is then 50000/10 = 5000. The results will be written into a file named `hac.out` in the same folder where you put your `run.in` file in.

12. The `run` keyword.

    This keyword only requires a single parameter, which is the number of steps for the current run. The time-evolution will only start when a `run` keyword has been reached. Before reaching this keyword, the code just collects the parameters for the current run. In the case where the VAC or the HAC is calculated, the number of steps should be larger than the product of the sampling interval and the number of correlation data. For example, the parameters in the commands

    ```
    compute_hac 10 100000 10
    run 10000000
    ```

are reasonably good because the number of steps (10^7) is 10 times as large as the product of the sampling interval and the number of correlation data ($10 \times 10^5 = 10^6$). In the case of calculating the VAC, it is important to first estimate the amount of memory to be used. Denote the number of steps as $N_{\rm run}$ and the sampling interval as $N_{\rm samp}$, the memory to be used for holding the velocity data is $(N_{\rm run}/N_{\rm samp}) \times N \times 3 \times 8$ bytes if using double-precision. If the number of atoms is $N = 10^4$, $N_{\rm samp} = 5$, and $N_{\rm run} = 10^5$, the memory to be used for holding the velocity data is about 4.8 GB. This is ok for Tesla K40 and K80, but may be too much for older GPUs.

5.3 Output files of GPUMD

In this section, we describe the formats of the output files. The output files and the corresponding keywords (used in the `run.in` file) generating them are:

- `thermo.out` is generated by `dump_thermo`
- `xyz.out` is generated by `dump_position`
- `velocity.out` is generated by `dump_velocity`
- `force.out` is generated by `dump_force`
- `potential.out` is generated by `dump_potential`
- `virial.out` is generated by `dump_virial`
- `vac.out` is generated by `compute_vac`
- `hac.out` is generated by `compute_hac`
- `shc.out` is generated by `compute_shc`
- `temperature.out` is generated by `compute_temp`

5.3.1 An important note

For all the output files, data from a new simulation will be appended to existing data. Therefore, if you do not intend to append new data to existing ones, you'd better first remove the existing output file or rename it.

5.3.2 The thermo.out file

This file is generated by using the `dump_thermo` keyword in the `run.in` file. There are 9 columns in the thermo.out file, each containing the values of a quantity at increasing time points. The quantities are as follows:

- column 1: temperature (in unit of K)
- column 2: total energy (in unit of eV) of the system if the thermostat method is Nosé-Hoover chain and kinetic energy (in unit of eV) of the system otherwise
- column 3: total energy (in unit of eV) of the thermostat if the thermostat method is Nosé-Hoover chain and potential energy (in unit of eV) of the system otherwise

- column 4: pressure (in unit of GPa) in the x direction
- column 5: pressure (in unit of GPa) in the y direction
- column 6: pressure (in unit of GPa) in the z direction
- column 7: box length (in unit of Å) in the x direction
- column 8: box length (in unit of Å) in the y direction
- column 9: box length (in unit of Å) in the z direction

5.3.3 The xyz.out file

There are 3 columns in the xyz.out file, corresponding to the x, y, and z coordinates of the system at increasing time points. For example, if there are 4 atoms (labelled from 0 to 3) and you have saved 2 frames (corresponding to t_0 and t_1) of the configuration into the `xyz.out` file, the data will be arranged in the following way:

```
x_0(t_0) y_0(t_0) z_0(t_0)
x_1(t_0) y_1(t_0) z_1(t_0)
x_2(t_0) y_2(t_0) z_2(t_0)
x_3(t_0) y_3(t_0) z_3(t_0)
x_0(t_1) y_0(t_1) z_0(t_1)
x_1(t_1) y_1(t_1) z_1(t_1)
x_2(t_1) y_2(t_1) z_2(t_1)
x_3(t_1) y_3(t_1) z_3(t_1)
```

5.3.4 The velocity.out file

Similar to the `xyz.out` file, but for the velocities, in the natural unit used in GPUMD.

5.3.5 The force.out file

Similar to the `xyz.out` file, but for the forces, in unit of eV/Å.

5.3.6 The potential.out file

Similar to the `xyz.out` file, but for the potentials in unit of eV. Note that there is only one column, as potential is a scalar.

5.3.7 The virial.out file

Similar to the `xyz.out` file, but for the virial stresses, in unit of eV. Note that the three columns correspond to the three diagonal elements of the virial stress tensor. The off-diagonal elements are not dumped.

5.3.8 The vac.out file

This file contains the data of VAC (velocity autocorrelation) and related quantities, namely, the RDC (running diffusion coefficient) and the PDOS (phonon density of states). The data in this file are organized as follows:

- column 1: correlation time (in unit of ps)
- column 2: VAC (in unit of $\text{Å}^2/\text{ps}^2$) in the x direction
- column 3: VAC (in unit of $\text{Å}^2/\text{ps}^2$) in the y direction
- column 4: VAC (in unit of $\text{Å}^2/\text{ps}^2$) in the z direction
- column 5: RDC (in unit of $\text{Å}^2/\text{ps}$) in the x direction
- column 6: RDC (in unit of $\text{Å}^2/\text{ps}$) in the y direction
- column 7: RDC (in unit of $\text{Å}^2/\text{ps}$) in the z direction
- column 8: angular frequency ω in unit of THz
- column 9: DOS (in unit of 1/THz) in the x direction
- column 10: DOS (in unit of 1/THz) in the y direction
- column 11: DOS (in unit of 1/THz) in the z direction

5.3.9 The hac.out file

This file contains the data of HAC (heat current autocorrelation) and RTC (running thermal conductivity), organized in the following way:

- column 1: correlation time (in unit of ps)
- column 2: $\langle J_x^{\text{in}}(0)J_x^{\text{in}}(t)\rangle$ (in unit of eV^3/amu)
- column 3: $\langle J_x^{\text{out}}(0)J_x^{\text{out}}(t)\rangle$ (in unit of eV^3/amu)
- column 4: $2\langle J_x^{\text{in}}(0)J_x^{\text{out}}(t)\rangle$ (in unit of eV^3/amu)
- column 5: $\langle J_y^{\text{in}}(0)J_y^{\text{in}}(t)\rangle$ (in unit of eV^3/amu)
- column 6: $\langle J_y^{\text{out}}(0)J_y^{\text{out}}(t)\rangle$ (in unit of eV^3/amu)
- column 7: $2\langle J_y^{\text{in}}(0)J_y^{\text{out}}(t)\rangle$ (in unit of eV^3/amu)
- column 8: $\langle J_z(0)J_z(t)\rangle$ (in unit of eV^3/amu)
- column 9: $\kappa_x^{\text{in}}(t)$ (in unit of $\text{Wm}^{-1}\text{K}^{-1}$)
- column 10: $\kappa_x^{\text{out}}(t)$ (in unit of $\text{Wm}^{-1}\text{K}^{-1}$)
- column 11: $\kappa_x^{\text{cross}}(t)$ (in unit of $\text{Wm}^{-1}\text{K}^{-1}$)
- column 12: $\kappa_y^{\text{in}}(t)$ (in unit of $\text{Wm}^{-1}\text{K}^{-1}$)

- column 13: $\kappa_y^{\rm out}(t)$ (in unit of $\rm Wm^{-1}K^{-1}$)
- column 14: $\kappa_y^{\rm cross}(t)$ (in unit of $\rm Wm^{-1}K^{-1}$)
- column 15: $\kappa_z(t)$ (in unit of $\rm Wm^{-1}K^{-1}$)

Note that the HAC and the RTC have been decomposed as described in Ref. [7]. This decomposition is useful for 2D materials but is not necessary for 3D materials. For 3D materials, one can sum up some columns to ge the conventional data. For example:

$$\langle J_x(0)J_x(t)\rangle = \langle J_x^{\rm in}(0)J_x^{\rm in}(t)\rangle + \langle J_x^{\rm out}(0)J_x^{\rm out}(t)\rangle + 2\langle J_x^{\rm in}(0)J_x^{\rm out}(t)\rangle. \tag{5.1}$$

$$\kappa_x(t) = \kappa_x^{\rm in}(t) + \kappa_x^{\rm out}(t) + \kappa_x^{\rm cross}(t). \tag{5.2}$$

5.3.10 The temperature.out file

This file contains data related to NEMD simulations of heat transport. Assuming that the system is divided into M groups, then

- columns 1 to M are the block temperatures
- the last second column is the total energy of the thermostat coupling to the heat source region
- the last column is the total energy of the thermostat coupling to the heat sink region

5.3.11 The shc.out file

This file contains data for the nonequilibrium heat flux correlation function $K_{A\to B}(t)$ as defined in Eq. (3.86) times the cross-section area. That is, one has to divide the data here by the cross-sectional area to get $K_{A\to B}(t)$. There are two columns, which are $K_{A\to B}^{\rm in}(t)$ and $K_{A\to B}^{\rm out}(t)$ defined in Ref. [7]. If one does not need the decomposition, one can sum up the two columns to get the total correlation function. Note that an individual correlation function corresponds to N_c consecutive rows.

Chapter 6

Examples

In this chapter, we give some examples to illustrate the usage of GPUMD. All the results presented here are obtained by using the **double-precision** version of the code. The single-precision version is faster but we are not sure whether it is always as safe to use. Note that even the double-precision version of GPUMD is highly efficient. For details of the performance evaluation, see Ref. [6].

For each example, we have provided a `run.in` file and two MATLAB scripts:

- `create_xyz.m`, which can be used to create the `xyz.in` input file
- `plot_results.m`, which can used to analyze the output data

After gaining some experiences in using GPUMD, one can use any other program to do these pre-processing and post-processing jobs.

6.1 Thermal expansion of silicon crystal

A given crystal should have a well defined average lattice constant at a given pressure and temperature. Here we use silicon as an example to show how to calculate lattice constants using GPUMD. We use a cubic system (of diamond structure) consisting of $10^3 \times 8 = 8000$ silicon atoms and use the Tersoff-1989 potential.

The first few lines of the `xyz.in` file created by the `create_xyz.m` MATLAB script are:

```
8000 4 3
1 1 1 54.3 54.3 54.3
0 0 28 0 0 0
0 0 28 0 2.715 2.715
0 0 28 2.715 0 2.715
0 0 28 2.715 2.715 0
```

The first line tells that the number of particles is 8000, the neighbor list size will be 8000×4, and the initial cutoff distance for the neighbor list construction is 3 Å. These parameters are good for silicon crystal described by the Tersoff potential, because no atom can have more than 4 neighbor atoms in the temperature range studied. One can make the second number larger, which only results in using more memory. If this numver is not large enough, GPUMD will give an error message and exit. Note that all the atom types and group labels are 0 in this simulation.

The "run.in" input file is given below. The first line of command tells that the potential to be used is specified in the file `potentials/si_tersoff_1989_1.txt`. The second line of the command tells that the velocities will be initialized with a temperature of 1 K. Then, the next 4 lines tell how to do the first run. This run will be in the NPT ensemble, using the Berendsen method. The temperature is 1 K and the pressures are zero in all the directions. The coupling constants are 0.01 (dimensionless) and 0.0005 (in the natural unit system adopted by GPUMD) for the thermostat and the barostat, respectively. The time step for integration is 1 fs. There are 10^5 steps for this run and the thermodynamic quantities will be output every 100 steps. After this run, there are 5 other runs with the same parameters but the target temperature. Note that the time step only needs to be set once if one is intended to use the same time step in the whole simulation. In contrast, one has to use the `dump_thermo` keyword for each run in order to get outputs for each run.

```
#-----------------------------------------------------------------
potential   potentials/si_tersoff_1989_1.txt
velocity    1

ensemble    npt_ber 1 1 0.01 0 0 0 0.0005
time_step   1
dump_thermo 100
run         100000

ensemble    npt_ber 200 200 0.01 0 0 0 0.0005
dump_thermo 100
run         100000

ensemble    npt_ber 400 400 0.01 0 0 0 0.0005
dump_thermo 100
run         100000

ensemble    npt_ber 600 600 0.01 0 0 0 0.0005
dump_thermo 100
run         100000

ensemble    npt_ber 800 800 0.01 0 0 0 0.0005
dump_thermo 100
run         100000

ensemble    npt_ber 1000 1000 0.01 0 0 0 0.0005
dump_thermo 100
run         100000
#-----------------------------------------------------------------
```

It takes about 4 min to run this example when a Tesla K40 card is use. The speed of the run is about 1.9×10^7 atom $\times$ step/second.

The output file `thermo.out` contains many useful data, which can be analyzed by the MATALB script `plot_results.m`. The results are shown in Fig. 6.1:

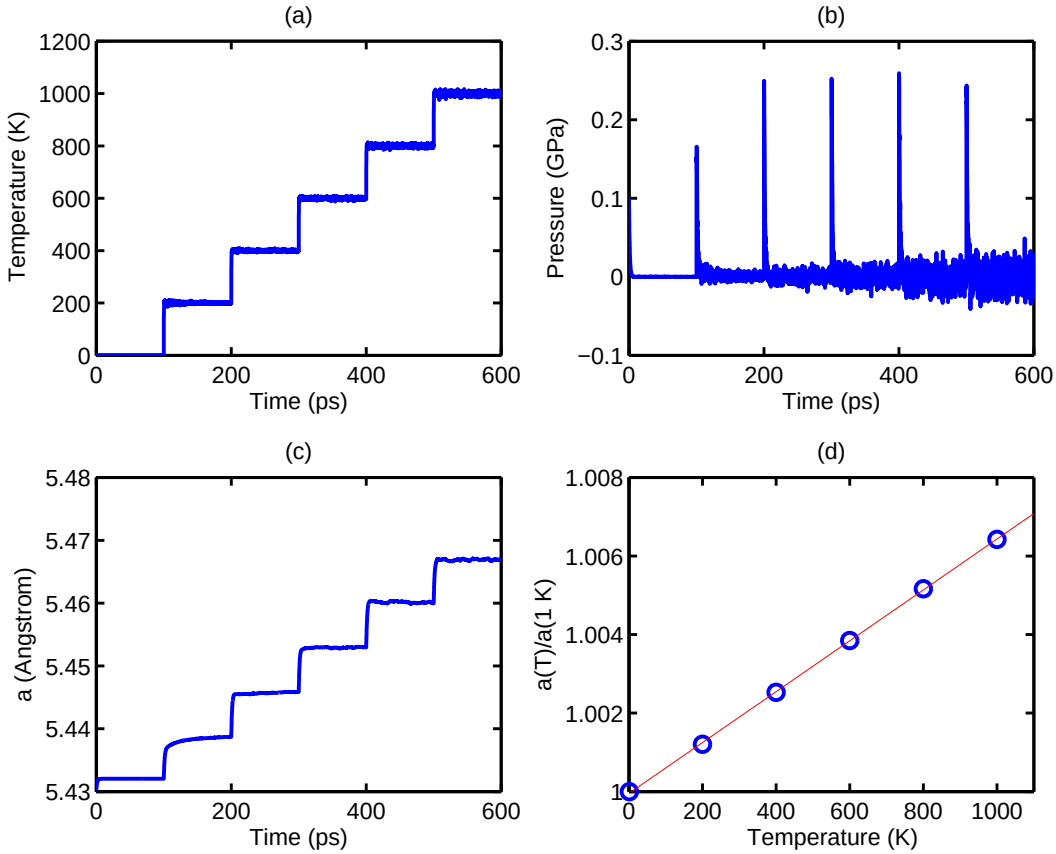

Figure 6.1: (a) Instant temperature as a function of simulations time. (b) Instant pressure as a function of simulation time. (c) Instant lattice constant as a function of simulations time. (d) Normalized average lattice constant (over the last 50 ps in each run for a given temperature) as a function of temperature.

- (a): The temperature for each run quickly reaches the target temperature (with fluctuations).
- (b): The pressure (averaged over the three directions) for each run quickly reaches the target pressure zero (with fluctuations).
- (c): The lattice constant (averaged over the three directions) for each run reaches a plateau (with fluctuations) after some steps.
- (d): We calculate the average lattice constant at each temperature by averaging the second half of the data for each run. The average lattice constants at different temperatures can be well fit by a linear function, with the thermal expansion coefficient being estimated to be $\alpha \approx 6.5 \times 10^{-6}$ K^{-1}.

6.2 Phonon density of states of graphene

In this example, we calculate the phonon density of states of graphene at 300 K and zero pressure. The simulated cell size is about 15 nm $\times$ 15 nm (8 640 atoms). The first few

lines of the `xyz.in` file are:

```
8640 3 2.1
1 1 0 149.649 155.52 3.35
0 0 12 1.24708 0 0
0 0 12 0 0.72 0
0 0 12 0 2.16 0
0 0 12 1.24708 2.88 0
```

This is a stable structure with 3 neighbors for each atom when periodic boundary conditions are applied in the planar directions (x and y). In the z direction, free boundary conditions are used. Every atom is of type 0 and in group 0.

The `run.in` file reads:

```
#-------------------------------------------------------------------
potential   potentials/c_tersoff_fan_2017.txt
velocity    300

ensemble    npt_ber 300 300 0.01 0 0 0 0.0005
time_step   1
dump_thermo 1000
run         1000000

ensemble    nve
compute_vac 5 200 400
run         100000

ensemble    nve
compute_vac 5 200 400
run         100000

ensemble    nve
compute_vac 5 200 400
run         100000

ensemble    nve
compute_vac 5 200 400
run         100000

ensemble    nve
compute_vac 5 200 400
run         100000
#-------------------------------------------------------------------
```

The potential model is Tersoff-1989, but some parameters are those reparameterized by Lindsay and Broido [17]. In the version by Lindsay and Broido, the carbon-carbon bond length at zero temperature is 1.44 Å, which is larger than the experimental value, 1.42 Å. Because the only relevant length parameters in the Tersoff-1989 potential are λ and μ in the repulsive and attractive functions, we can simply correct the bond length by

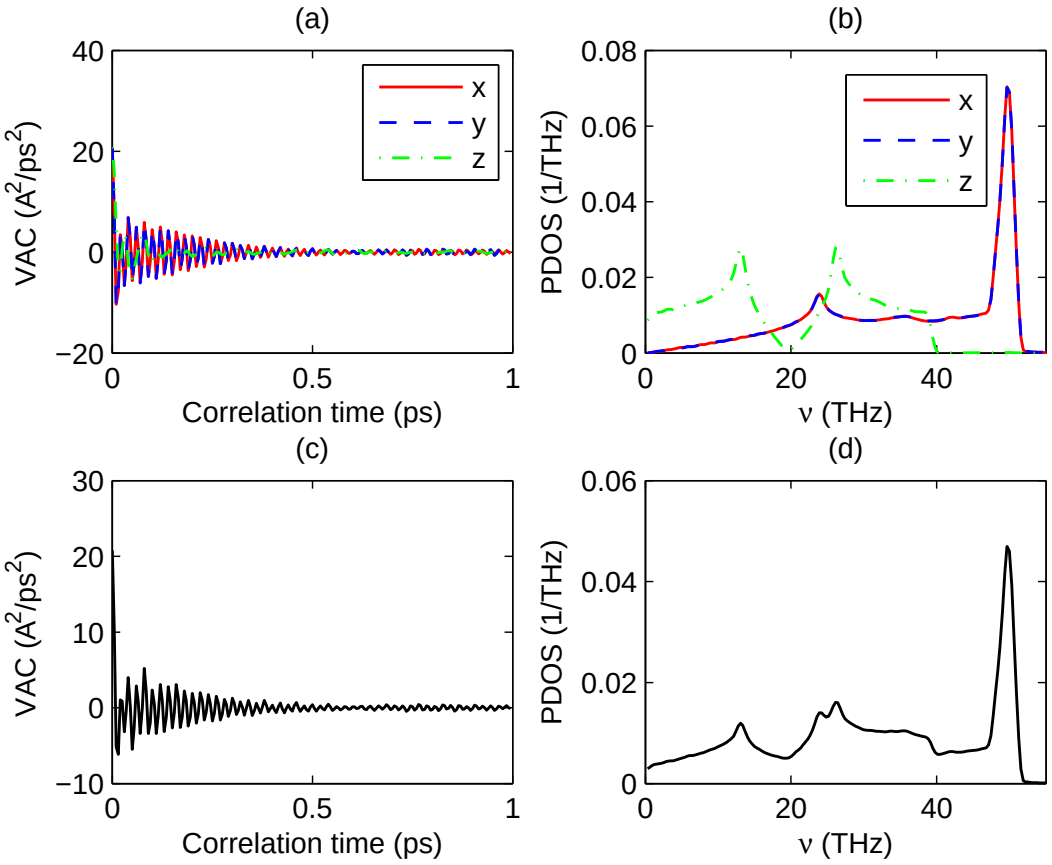

Figure 6.2: (a) VAC as a function of correlation time for the separate directions. (b) PDOS as a function of the phonon frequency for the separate directions. (c) VAC as a function of correlation time averaged over the separate directions. (d) PDOS as a function of the phonon frequency average over the separate directions.

a proper scaling of these two parameters. All the other parameters are not affected by this scaling.

There are 6 runs. The first run serves as the equilibration stage, where the NPT ensemble is used. This run lasts 1 ns. The other 5 runs are identical production runs. In each production run, the NVE ensemble is used. The line with `compute_vac` means that velocities will be recorded every 5 steps (5 fs) and 200 VAC data (the maximum correlation time is then about 1 ps) will be calculated. The last parameter in this line is the maximum angular frequency considered, $\omega_{\max} = 2\pi\nu_{\max} = 400$ THz, which is large enough for graphene. Each production run lasts 100 ps. The major reason for using multiple production runs rather a single one is that computing the VAC requires a lot of memory, which prevents using very long runs.

This simulation takes about 7 min when a Tesla K40 is used. The speed of this simulation, being about 3.6×10^7 atom $\times$ step/second, is higher than that of the previous example because the number of neighbors for each atom is smaller here (numbers of atoms are comparable and the potential models are the same).

Figure 6.2 shows the calculated VAC and PDOS. For 3D isotropic systems, the results along different directions are equivalent and can be averaged, but for 2D materials

like graphene, it is natural to consider the in-plane part (the x and y directions in the simulation) and the out-of-plane part (the z direction) separately. It can be seen that the two components behave very differently. We can see that the cutoff frequency for the out-of-plane component (~ 40 THz) is smaller than that for the in-plane component (~ 52 THz), which means that the two components have different Debye temperatures.

6.3 Thermal conductivity of graphene

In this example, we use the Green-Kubo method to calculate the lattice thermal conductivity of graphene at 300 K and zero pressure. The `xyz.in` file and the potential parameters used are the same as in the last example. Note that the thickness of the graphene sheet is set to 3.35 Å according to the convention in the literature. This thickness is needed to calculate an effective 3D thermal conductivity for a 2D material.

The `run.in` file for this simulation reads:

```
#------------------------------------------------------------------
potential           potentials/c_tersoff_fan_2017.txt
velocity            300

ensemble            npt_ber 300 300 0.01 0 0 0 0.0005
time_step           1
dump_thermo         1000
run                 1000000

ensemble            nve
compute_hac         20 50000 10
run                 10000000
#------------------------------------------------------------------
```

The equilibrium stage is the same as in the last example. In the production stage, we use the NVE ensemble and calculate the HAC (heat current autocorrelation) and RTC (running thermal conductivity). The sampling interval is 20, the number of correlation steps is 50000 (such that the maximum correlation time is about 10^6 fs = 1 ns), and the HAC and RTC are averaged for every 10 data points before written out. The production time is 10 ns, which is 10 times as long as the maximum correlation time. This is a reasonable choice.

It takes about one hour to complete the simulation using a Tesla K40 card. Figure shows the results from a single simulation. Note that the output file `hac.out` contain decomposed HACs and RTCs as described in Ref. [7]. As the system is essentially isotropic, we can average over the two directions. The results are presented in Fig. 6.3. From (a), we can see that the in-plane component and the out-of-plane component have different time scales. The latter decays much more slowly. In (b), the RTC components κ^{in}, κ^{out}, κ^{cross}, together with the RTC in the z direction κ_z, are presented. It is clear that κ^{out} converges much more slowly than κ^{in} and to a larger magnitude. This is an accepted result stating that heat transport in suspended pristine graphene is dominated by the flexural (out-of-plane) phonons. Note that heat transport in the z direction here is meaningless because free boundary conditions are applied in this direction. However, the fact that the calculated κ_z should converged to zero can provide a validation of the results.

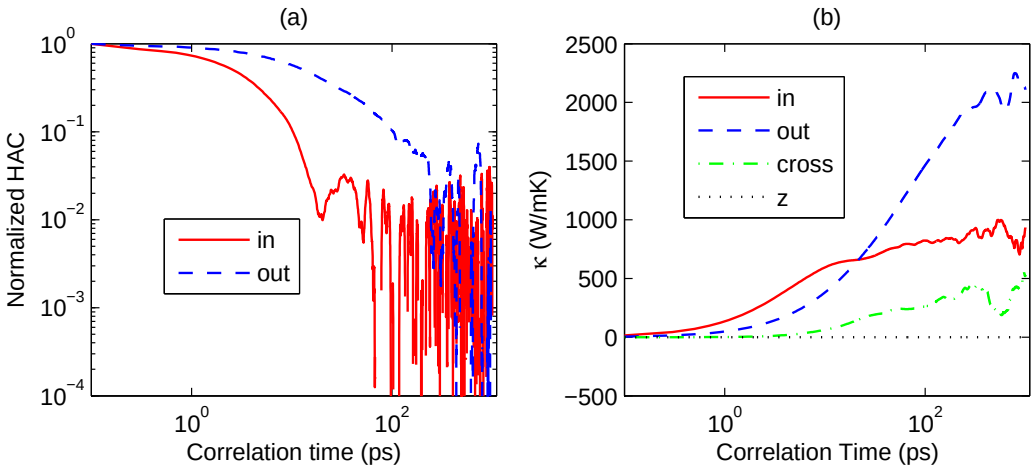

Figure 6.3: (a) Normalized HAC as a function of correlation time for the in-plane and out-of-plane components. (b) RTC as a function of correlation time for various components.

Accurately calculating thermal conductivity of graphene can be a very time consuming task. The results we presented are from a single simulation with a production time of 10 ns. It can been seen that the data already becomes very noisy when the correlation time is 100 ps. To obtain accurate results, one needs to do many independent simulations and do a statistical average. Much more accurate data were presented in Fig. 2 of Ref. [7]. Here are the simulation parameters used in Ref. [7] which differ from those used in this example:

- The simulation cell size used in Ref. [7] is larger, which is about 25 nm × 25 nm (24000 atoms).
- The maximum correlation time used in Ref. [7] is larger, which is 10 ns.
- The production time used in Ref. [7] for one independent simulation is larger, which is 50 ns.
- There are 100 independent simulations in Ref. [7], not a single one here.

Each independent simulation in Ref. [7] took about 10 GPU hours (using Tesla K40) and about 1000 GPU hours were used to obtain the results shown in Fig. 2 of Ref. [7].

6.4 Ballistic thermal conductance of graphene

In this example, we show how to study heat transport using the NEMD method combined with the spatial and spectral decompositions as described in Ref. [7]. We aim to obtain similar results for the case of unstrained graphene as presented in Fig. 4 of Ref. [7].

In the NEMD simulation, periodic boundary conditions are applied to the transverse direction (chosen as the zigzag direction) and fixed boundary conditions are applied to the transport direction (chosen as the armchair direction). The width of the simulated system is about 10 nm and the total length in the transport direction is about 70 nn. One major difference from the previous simulations is that here the group labels are not identically 0. We divide the system into 8 groups along the transport direction and label

the groups from 1 to 8. Groups and 1 and 8 are taken as the source and sink regions, respectively. The number of atoms in groups 1 to 8 are 8000, 1600, 1600, 1600, 1600, 1600, 1600, and 8000, respectively. Some extra fixed atoms are put into group 0. Here are two important notes on the group label:

- It starts from 0.
- In the `xyz.in` file, an atom with smaller group label should appear earlier than that with a larger group label. We may remove this restriction in a future version but one should remember this rule when using the current version.

The `run.in` file for this example reads:

```
#-----------------------------------------------------------------
potential    potentials/c_tersoff_fan_2017.txt
velocity     300

ensemble     nvt_ber 300 300 0.01
fix          0
time_step    1
dump_thermo  1000
run          1000000

ensemble     heat_nhc 300 100 10 1 8
fix          0
compute_temp 1000
compute_shc  2 250 100000 4 5
run          2000000
#-----------------------------------------------------------------
```

In this simulation, we fix the lattice constant and only control the temperature in the equilibration stage. The `fix 0` command is used to realize the fixed boundary conditions by fixing the atoms in group 0. In the production stage, the `heat_hac` "ensemble" type is used to generate the nonequilibrium heat flux. The heat source (group 1) and the heat sink (group 8) will be maintained at 310 K and 290 K, respectively. The block temperatures will be output every 1000 integration steps. The command with the keyword `compute_shc` means that the nonequilibrium heat flux correlation function defined in Eq. (3.86) will be computed: the sampling interval is 2, the number of correlation steps is 250, the number of steps for calculating one correlation function is 100000, and the heat flux considered flows from group 4 to group 5 (the middle interface of the simulated system).

This simulation takes about 40 min using a Tesla K40 card. Figure 6.4 shows the results obtained from the `temperature.out` file:

- (a): The block temperatures show a relatively smooth profile, but no clear linear region can be identified. Actually, heat transport here is ballistic and we do not expect to find a well defined temperature gradient (which is need for computing the thermal conductivity). What is important is that a well defined temperature difference (20 K), which is needed for computing the ballistic thermal conductance, can be established.

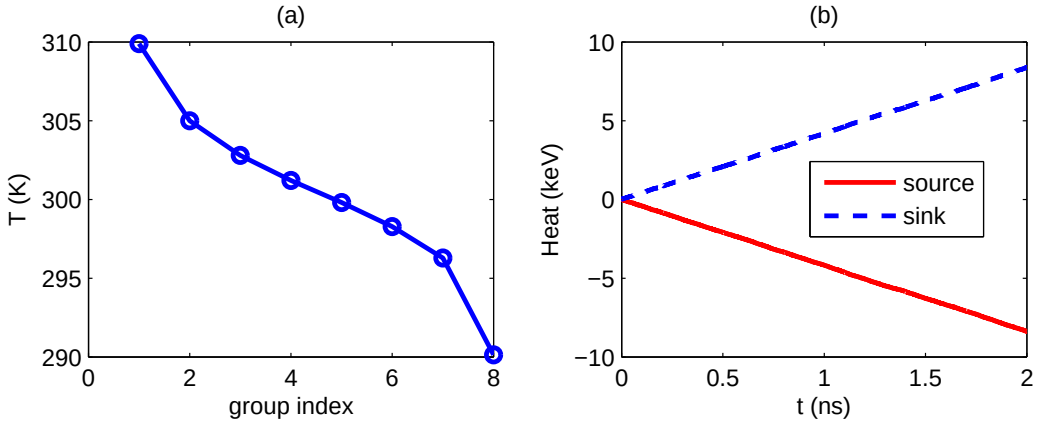

Figure 6.4: (a) Temperature profile in the NEMD simulation. (b) Total energy of the thermostats as a function of simulation time in the production stage.

- (b): A steady energy exchange between the system and the thermostats in the source and sink regions has been well established. The conequilibrium heat current can be estimated to be about $Q = 4.2$ eV/ps. Then a ballistic conductance of about $G = 10.2$ GW m^{-2} K^{-1} can be obtained. This classical value overestimates the correct one and we will add discussion about quantum corrections after some of our submitted manuscripts get published.

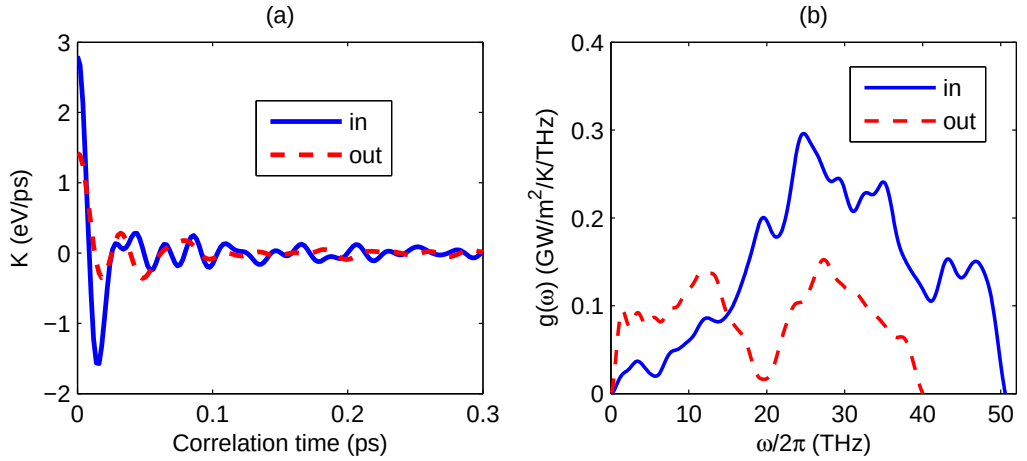

Figure 6.5: (a) The nonequilibrium heat flux autocorrelation function as define in Eq. (3.86) as a function of correlation time. (b) The spectrally decomposed ballistic conductance as a function of phonon frequency.

The `shc.out` contains data for the nonequilibrium heat flux autocorrelation function $K(t)$ as define in Eq. (3.86). Also, the in-out decompositions introduced in Ref. [7] is considered. The calculated $K(t)$ and the spectrally decomposed conductance $g(\omega)$ for the in-plane and the out-of-plane components are shown in Fig. 6.5. One can see that they are similar to the velocity autocorrelation and the phonon density of states discussed in a previous example. This is reasonable because the ballistic conductance is proportional to the product of the phonon density of states and the group velocity.

One can check the consistency of the results at least in the following two ways:

- When steady state is achieved, the correlation function $K(t)$ evaluated at zero correlation time should be consistent with the heat current calculated from the energy exchange rate between the system and the thermostats. That is, $K(0) = Q$.
- The total thermal conductance G should equal the integration of the spectral conductance. That is, $G = \int_0^\infty \frac{d\omega}{2\pi} g(\omega)$.

One should always make sure that the obtained data pass these tests approximately.

Bibliography

[1] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak. Molecular dynamics with coupling to an external bath. *The Journal of Chemical Physics*, 81(8):3684–3690, 1984.

[2] Youping Chen and Adrian Diaz. Local momentum and heat fluxes in transient transport processes and inhomogeneous systems. *Phys. Rev. E*, 94:053309, Nov 2016.

[3] X D Dai, Y Kong, J H Li, and B X Liu. Extended finnis-sinclair potential for bcc and fcc metals and alloys. *Journal of Physics: Condensed Matter*, 18(19):4527, 2006.

[4] Murray S. Daw and M. I. Baskes. Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals. *Phys. Rev. B*, 29:6443–6453, Jun 1984.

[5] J. M. Dickey and A. Paskin. Computer simulation of the lattice dynamics of solids. *Phys. Rev.*, 188:1407–1418, Dec 1969.

[6] Zheyong Fan, Wei Chen, Ville Vierimaa, and Ari Harju. Efficient molecular dynamics simulations with many-body potentials on graphics processing units. *Computer Physics Communications*, 218:10 – 16, 2017.

[7] Zheyong Fan, Luiz Felipe C. Pereira, Petri Hirvonen, Mikko M. Ervasti, Ken R. Elder, Davide Donadio, Tapio Ala-Nissila, and Ari Harju. Thermal conductivity decomposition in two-dimensional materials: Application to graphene. *Phys. Rev. B*, 95:144309, Apr 2017.

[8] Zheyong Fan, Luiz Felipe C. Pereira, Hui-Qiong Wang, Jin-Cheng Zheng, Davide Donadio, and Ari Harju. Force and heat current formulas for many-body potentials in molecular dynamics simulations with applications to thermal conductivity calculations. *Phys. Rev. B*, 92:094301, Sep 2015.

[9] Zheyong Fan, Topi Siro, and Ari Harju. Accelerated molecular dynamics force evaluation on graphics processing units for thermal conductivity calculations. *Computer Physics Communications*, 184(5):1414 – 1425, 2013.

[10] Christopher J. Fennell and J. Daniel Gezelter. Is the ewald summation still necessary? pairwise alternatives to the accepted standard for long-range electrostatics. *The Journal of Chemical Physics*, 124(23):234104, 2006.

[11] Melville S. Green. Markoff random processes and the statistical mechanics of time-dependent phenomena. ii. irreversible processes in fluids. *The Journal of Chemical Physics*, 22(3):398–413, 1954.

[12] Robert J. Hardy. Atomistic formulas for local properties in systems with many-body interactions. *The Journal of Chemical Physics*, 145(20):204103, 2016.

[13] William G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Phys. Rev. A*, 31:1695–1697, Mar 1985.

[14] T. Ikeshoji and B. Hafskjold. Non-equilibrium molecular dynamics calculation of heat conduction in liquid and through liquid-gas interface. *Molecular Physics*, 81(2):251–261, 1994.

[15] Philippe Jund and Rémi Jullien. Molecular-dynamics calculation of the thermal conductivity of vitreous silica. *Phys. Rev. B*, 59:13707–13711, Jun 1999.

[16] Ryogo Kubo. Statistical-mechanical theory of irreversible processes. i. general theory and simple applications to magnetic and conduction problems. *Journal of the Physical Society of Japan*, 12(6):570–586, 1957.

[17] L. Lindsay and D. A. Broido. Optimized tersoff and brenner empirical potential parameters for lattice dynamics and phonon thermal transport in carbon nanotubes and graphene. *Phys. Rev. B*, 81:205441, May 2010.

[18] G. J. Martyna, M. E. Tuckerman, D. J. Tobias, and M. L. Klein. Explicit reversible integrators for extended systems dynamics. *Mol. Phys.*, 87:1117, 1996.

[19] Glenn J. Martyna, Michael L. Klein, and Mark Tuckerman. Nosé-hoover chains: The canonical ensemble via continuous dynamics. *The Journal of Chemical Physics*, 97(4):2635–2643, 1992.

[20] Florian Müller-Plathe. A simple nonequilibrium molecular dynamics method for calculating the thermal conductivity. *The Journal of Chemical Physics*, 106(14):6082–6085, 1997.

[21] Shuichi Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of Chemical Physics*, 81(1):511–519, 1984.

[22] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117:1, 1995.

[23] K. Sääskilahti, J. Oksanen, J. Tulkki, and S. Volz. Role of anharmonic phonon scattering in the spectrally decomposed thermal conductance at planar interfaces. *Phys. Rev. B*, 90:134312, Oct 2014.

[24] K. Sääskilahti, J. Oksanen, S. Volz, and J. Tulkki. Frequency-dependent phonon mean free path in carbon nanotubes from nonequilibrium molecular dynamics. *Phys. Rev. B*, 91:115426, Mar 2015.

[25] Frank H. Stillinger and Thomas A. Weber. Computer simulation of local order in condensed phases of silicon. *Phys. Rev. B*, 31:5262–5271, Apr 1985.

[26] William C. Swope, Hans C. Andersen, Peter H. Berens, and Kent R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 76(1):637–649, 1982.

[27] J. Tersoff. Modeling solid-state chemistry: Interatomic potentials for multicomponent systems. *Phys. Rev. B*, 39:5566–5568, Mar 1989.

[28] J. Tersoff. Erratum: Modeling solid-state chemistry: Interatomic potentials for multicomponent systems. *Phys. Rev. B*, 41:3248–3248, Feb 1990.

[29] Mark E. Tuckerman. *Statistical Mechanics: Theory and Molecular Simulation*. Oxford University Press, 2010.

[30] D. Wolf, P. Keblinski, S. R. Phillpot, and J. Eggebrecht. Exact method for the simulation of coulombic systems by spherically truncated, pairwise r1 summation. *The Journal of Chemical Physics*, 110(17):8254–8282, 1999.

[31] X. W. Zhou, R. A. Johnson, and H. N. G. Wadley. Misfit-energy-increasing dislocations in vapor-deposited CoFe/NiFe multilayers. *Phys. Rev. B*, 69:144113, Apr 2004.

[32] Yanguang Zhou and Ming Hu. Quantitatively analyzing phonon spectral contribution of thermal conductivity based on nonequilibrium molecular dynamics simulations. ii. from time fourier transform. *Phys. Rev. B*, 92:195205, Nov 2015.