

GPUMD:
Graphics Processing Units
Molecular Dynamics

Reference Manual

Version 1.0

(Dated: October 25, 2016)

Authors:

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

Preface

This is the reference manual of GPUMD (version 1.0), a molecular dynamics simulation code implemented fully on graphics processing units (GPUs). GPUMD stands for **G**raphics **P**rocessing **U**nits **M**olecular **D**ynamics.

One of the major features of GPUMD is that force evaluation for many-body potentials, such as the Tersoff potential and the Stillinger-Weber potential, has been significantly accelerated using GPUs. The major effort we would like to make in the future is to implement many other important many-body potentials, such as the EAM/MEAM potentials and the Brenner/REBO/LCBOP potentials.

You can send an email to the first author (zheyong.fan@aalto.fi) if you find errors in the manual or bugs in the source code, or have any question about the manual and code.

Note that you don't need to read the whole manual before starting to use GPUMD. If you are not interested in the theoretical formalisms and algorithms, you can safely skip chapter 2. Only chapters 3 and 4 are essential to understanding how to use GPUMD.

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.

Contents

1	Features and non-features of GPUMD	1
1.1	GPU-accelerated force evaluation for many-body potentials	1
1.2	Utilities for computing thermal conductivity and related quantities	1
1.3	Other MD features and non-features of GPUMD	2
1.3.1	Neighbor list construction	2
1.3.2	Box and boundary conditions	2
1.3.3	Integrators and thermostats/barostats	2
2	Theoretical formalisms and algorithms	3
2.1	Units	3
2.2	Overall structure of the code	3
2.3	Neighbour list construction	4
2.3.1	The simple $O(N^2)$ -scaling method	4
2.3.2	A fast $O(N)$ -scaling method	4
2.4	Force evaluation and related calculations	5
2.4.1	Force	5
2.4.2	Virial and pressure	8
2.4.3	Heat current	9
2.5	Integration by one step	11
2.5.1	The NVE ensemble and the velocity-Verlet algorithm	11
2.5.2	Berendsen thermostat and barostat	13
2.5.3	Nosé-Hoover chain thermostat	14
2.6	Postprocess	17
2.6.1	Heat current autocorrelation and lattice thermal conductivity	17
2.6.2	Velocity autocorrelation and related quantities	18
3	How to use GPUMD	23
3.1	Compile GPUMD	23
3.2	Run simulations with GPUMD	23
3.2.1	The xyz.in input file	23
3.2.2	The run.in input file	24
3.2.3	Prepare a potential file	27
3.2.4	Prepare a “driver input file”	28
3.2.5	Run the code	28
3.3	Data formats in the output files	28
3.3.1	The thermo.out file	28
3.3.2	The xyz.out file	29
3.3.3	The v.out file	29
3.3.4	The f.out file	29
3.3.5	The vac.out file	29

3.3.6	The hac.out file	30
4	Examples	31
4.1	Lattice constant of silicon	31
4.2	Phonon density of states of graphene	32
4.3	Thermal conductivity of graphene	34
5	Publications that are related to GPUMD	35

Chapter 1

Features and non-features of GPUMD

1.1 GPU-accelerated force evaluation for many-body potentials

The major advantage of GPUMD over other public molecular dynamics (MD) packages is that force evaluation for many-body potentials, such as the Tersoff potential [1] and the Stillinger-Weber potential [2], has been significantly accelerated by using GPUs. Our efficient and flexible GPU-implementation of the force evaluation for many-body potentials relies on a new force expression derived in Ref. [3]. Using the new formula, the resulting GPU-implementation is nearly identical to the CPU implementation. See Ref. [4] for details of the performance evaluation.

The current version of GPUMD only uses one GPU card in a single run. We are currently working on multi-GPU extensions. Every functionality in GPUMD has a pure-CPU version as well as a GPU version. Therefore, if you don't have GPUs, you can also use the pure-CPU version of the code. The pure-CPU version is serial and is mostly used for the purpose of validation.

1.2 Utilities for computing thermal conductivity and related quantities

GPUMD has mostly been used to study heat transport so far. It has very handy commands to calculate thermal conductivity and related quantities.

In the current version of GPUMD, we have implemented the Green-Kubo method for computing the lattice thermal conductivity. In this method, one first computes the heat current autocorrelation (HAC) and then calculates the lattice thermal conductivity using the Green-Kubo formula. This is also called the equilibrium MD method because the heat current is calculated in equilibrium states. In Ref. [3], it has been shown that the (microscopic) heat current formula for many-body potentials differs from that for two-body potentials. The correct heat current formula has been implemented in GPUMD.

A related quantity is phonon density of states (PDOS), which can be computed from velocity autocorrelation function (VAC) by discrete Fourier transform. The same VAC can also be used to compute diffusion constant using the Green-Kubo formula, which is equivalent to that obtained from the mean square displacement (MSD) using Einstein's relation.

We are currently working on implementing other methods for heat transport calculations, such as the direct method (the nonequilibrium MD method) and methods for spectral decomposition.

1.3 Other MD features and non-features of GPUMD

1.3.1 Neighbor list construction

GPUMD has a simple $O(N^2)$ Verlet-list method as well as a fast $O(N)$ cell-list method for neighbour list construction. The $O(N^2)$ version is slow for large systems but is simpler to implement and can be used for validating more advanced implementations. The $O(N)$ version is much faster for large systems and is the default version used in GPUMD.

Note that in the current version, we have only considered systems with fixed topology, where the neighbor list does not need to be updated. We are still working on improving the data structures and functions (kernels) related to the neighbor list. The next version of GPUMD will be able to simulate processes involving melting, diffusion, or fracture.

1.3.2 Box and boundary conditions

The current version of GPUMD only supports rectangular simulation box with periodic or open (free) boundary conditions in each direction.

The minimum image convention is used in neighbor list construction and force evaluation in directions with periodic boundary conditions.

1.3.3 Integrators and thermostats/barostats

The velocity-Verlet [5] integration scheme is used for all the statistical ensembles.

Apart from the *NVE* ensemble (microscopic ensemble), GPUMD also supports the *NVT* ensemble (canonical ensemble) and the *NPT* ensemble (isothermal-isobaric ensemble).

For the *NVT* ensemble, two methods are implemented: the Berendsen weak-coupling thermostat [6] and the Nosé-Hoover chain thermostat [7–9].

For the *NPT* ensemble, only the Berendsen weak-coupling barostat [6] is implemented. We are currently working on implementing the MTTK (Martyna, Tuckerman, Tobias, and Klein) [10–12] method.

Chapter 2

Theoretical formalisms and algorithms

This chapter assumes that the reader is familiar with CUDA programming [14] and is more technically involved than the other chapters. Skipping this chapter does not prevent using the code correctly and fluently. If you are not interested in the details of the formalisms and algorithms, you can move on to the next chapter.

2.1 Units

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 amu^{1/2} eV^{-1/2}, which is about 10 fs
2. Velocity: eV^{1/2} amu^{-1/2}, which is about 0.1 Å/fs = 10 km/s
3. Force: eV/Å
4. Boltzmann's constant: $k_B \approx 0.863 \times 10^{-4}$ eV/K
5. Electrostatic constant: $k_C = \frac{1}{4\pi\epsilon_0} \approx 14.4$ eV Åe⁻²

Note that the input and output files do not necessary adopt these units. For example, pressure (stress) in both input and output files is in unit of GPa, rather than eV Å⁻³.

2.2 Overall structure of the code

Taking the GPU version as an example, the overall structure of the code can be expressed as follows:

1. Initialization:

- (a) Reading in input data from files and allocate memory on the CPU and GPU.
 - (b) Initializing positions, velocities, neighbour list, and forces.
2. Evolving the system by a number of steps. Each step consists of the following possible calculations:
 - (a) Updating the neighbour list with a frequency according to the input.
 - (b) Integrating the equations of motion by one step according to the ensemble type and other external conditions.
 - (c) Outputting data to files or save them to memory for later postprocessing.
 3. Postprocessing, including
 - (a) VAC calculation.
 - (b) HAC calculation.
 4. Repeating the above two steps as many times as you want.
 5. Finalizing: free the memory used.

2.3 Neighbour list construction

2.3.1 The simple $O(N^2)$ -scaling method

Algorithm 1 presents the pseudo code of constructing the Verlet neighbour list, which has an $O(N^2)$ scaling with respect to the number of atoms N .

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. These tricks apply to many of the other kernels.

2.3.2 A fast $O(N)$ -scaling method

This method partitions the system into cells and only searches for neighbours in the closest cells to avoid unnecessary comparisons between particles that are far away from each other. The cells are identical rectangular cubes with edge length of r_c , the neighbour cutoff distance. In case the system size is not a multiple of r_c in one or more dimensions, the outermost cells in those dimensions are made larger such that the system size is filled. This choice of cell dimensions is optimal, because it is the minimal size that guarantees that all neighbours of particle residing in cell j can be found in the neighbouring cells of j .

The pseudo-code of this method is outlined in Algorithm 2. The steps of the algorithm have been described in more detail in Algorithms 3 - 6, together with the algorithm for calculating the cell index for a given particle. The algorithms are presented in serial way for clarity, but they are ready to be parallelized by replacing the outermost for-loops with kernel calls similar to Algorithm 1. For the cumulative sum, we have used the `thrust::exclusive_scan` function from the thrust library.

At first glance it may seem that Algorithm 5 has some redundant calculation, since the count of particles in cell j is calculated twice. It is however necessary, because of the way we store the cell contents in a single array of length N . Even with doing some extra calculation and despite using the atomic operations, the time used for this part is negligible compared to the total time of neighbour list construction.

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 neighbours for particle i
Require: NL_{ik} is the index of the k th neighbour of particle i
Require: \mathbf{r}_i is the position vector of particle i

```

1:  $k \leftarrow 0$ 
2: if  $i < N$  then
3:   load  $\mathbf{r}_i$  from the global memory
4:   for  $j = 0$  to  $N - 1$  do
5:     if  $j = i$  then
6:       Continue
7:     end if
8:     load  $\mathbf{r}_j$  from the global memory and calculate  $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ 
9:     apply the minimum image convention to  $\mathbf{r}_{ij}$ 
10:    if  $|\mathbf{r}_{ij}|^2 < r_c^2$  then
11:       $NL_{ik} \leftarrow j$ 
12:       $k \leftarrow k + 1$ 
13:    end if
14:  end for
15:   $NN_i \leftarrow k$ 
16: end if

```

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

- 1: Determine number of cells
 - 2: Construct a list of particles contained in each cell
 - 3: Construct the neighbour list based on this list
-

2.4 Force evaluation and related calculations

2.4.1 Force

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

$$U = \sum_{i=1}^N U_i. \quad (2.1)$$

The site energy 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.

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

$$U_i = \frac{1}{2} \sum_{j \neq i} U_{ij}(r_{ij}). \quad (2.2)$$

Here, $r_{ij} = |\mathbf{r}_j - \mathbf{r}_i|$ is the distance between particles i and j and $U_{ij}(r_{ij})$ is the pair potential

Algorithm 3 Determine number of cells

Require: L_d is the length of simulation box in dimension d **Require:** r_c is the neighbour cutoff distance**Require:** N_{cells} is the total number of cells in the system1: **for** $d = \{x, y, z\}$ **do**2: $N_d \leftarrow \lfloor L_d/r_c \rfloor$ 3: **end for**4: $N_{cells} \leftarrow N_x N_y N_z$

Algorithm 4 Calculate cell index from \mathbf{r}_i

Require: i is the index of the particle**Require:** \mathbf{r}_i is the position vector of particle i **Require:** r_c neighbour cutoff distance**Require:** N_d is the number of cells in dimension d **Require:** I is the index of the cell i belongs to1: **for** d in $\{x, y, z\}$ **do**2: $I_d \leftarrow \lfloor r_i^{(d)}/r_c \rfloor$ 3: **if** $I_d \geq N_d$ **then**4: $I_d \leftarrow I_d - N_d$ 5: **end if**6: **if** $I_d < 0$ **then**7: $I_d \leftarrow I_d + N_d$ 8: **end if**9: **end for**10: $I \leftarrow I_x + I_y N_x + I_z N_x N_y$

Algorithm 5 Cell content construction

Require: \mathbf{r}_i is the position vector of particle i **Require:** C_j is the count of particles in cell j **Require:** S_j is the cumulative sum of C_j 1: Initialize $C_j = 0$ 2: **for** $i = 0$ to $N - 1$ **do**3: Calculate cell index j from \mathbf{r}_i 4: Atomic operation: $C_j \leftarrow C_j + 1$ 5: **end for**6: **for** $i = 0$ to $N_{cells} - 1$ **do**7: Calculate $S_j = \sum_{0,j-1} C_j$ 8: **end for**9: Initialize $C_j = 0$ 10: **for** $i = 0$ to $N - 1$ **do**11: Calculate cell index j from \mathbf{r}_i 12: $CC[S_j + C_j] \leftarrow i$ 13: Atomic operation: $C_j \leftarrow C_j + 1$ 14: **end for**

Algorithm 6 Neighbour list construction

Require: N_{cells} is the total number of cells
Require: C_j is the number of particles in cell j
Require: S_j is the cumulative sum of C_j
Require: CC is the array of cell contents
Require: r_c is neighbour cutoff distance
Require: NL_{am} is the m th neighbour of particle a
Require: NN_a is the number of neighbours for particle a

```

1: for  $j = 0$  to  $N_{cells} - 1$  do
2:   for  $n_1 = 0$  to  $C_j - 1$  do
3:      $a \leftarrow CC[S_j + n_1]$ 
4:      $m \leftarrow 0$ 
5:     for  $k$  in cells neighbouring  $j$  do
6:       for  $n_2 = 0$  to  $C_k - 1$  do
7:          $b \leftarrow CC[S_k + n_2]$ 
8:          $\mathbf{r}_{ba} = \mathbf{r}_b - \mathbf{r}_a$ 
9:         apply the minimum image convention to  $\mathbf{r}_{ba}$ 
10:        if  $|\mathbf{r}_{ba}|^2 < r_c^2$  then
11:           $NL_{am} \leftarrow j$ 
12:           $m \leftarrow m + 1$ 
13:        end if
14:      end for
15:    end for
16:     $NN_a \leftarrow m$ 
17:  end for
18: end for

```

between them. The total force acted on particle i can be derived to be:

$$\mathbf{F}_i = -\nabla_i U = \sum_{j \neq i} \frac{\partial U_{ij}(r_{ij})}{\partial r_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}}. \quad (2.3)$$

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

$$\boxed{\mathbf{r}_{ij} \equiv \mathbf{r}_j - \mathbf{r}_i}. \quad (2.4)$$

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:

$$\mathbf{F}_i = \sum_{j \neq i} \mathbf{F}_{ij}, \quad (2.5)$$

where

$$\mathbf{F}_{ij} = \frac{\partial U_{ij}(r_{ij})}{\partial r_{ij}} \frac{\mathbf{r}_{ij}}{r_{ij}} \quad (2.6)$$

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

$$\mathbf{F}_{ij} = -\mathbf{F}_{ji}. \quad (2.7)$$

In some many-body potentials such as the embedded-atom potential [13], the site potential can not be written in the form of Eq. (2.2). In some other many-body potentials such as the

Tersoff potential, the site potential can be written in the form of Eq. (2.2), 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 respects Newton's third law explicitly has been derived as [3]:

$$\mathbf{F}_i = \sum_{j \neq i} \mathbf{F}_{ij} \quad (2.8)$$

where

$$\mathbf{F}_{ij} = -\mathbf{F}_{ji} = \frac{\partial U_i}{\partial \mathbf{r}_{ij}} - \frac{\partial U_j}{\partial \mathbf{r}_{ji}}. \quad (2.9)$$

2.4.2 Virial and pressure

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 ¹

$$\mathbf{W} = \sum_i \mathbf{W}_i, \quad (2.10)$$

$$\mathbf{W}_i = \mathbf{r}_i \otimes \mathbf{F}_i. \quad (2.11)$$

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

$$\mathbf{W}_i = -\frac{1}{2} \sum_{j \neq i} \mathbf{r}_{ij} \otimes \mathbf{F}_{ij}, \quad (2.12)$$

where only relative positions \mathbf{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:

$$p_{\text{ideal}} = \frac{Nk_B T}{V}, \quad (2.13)$$

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

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

$$\sigma^{\alpha\beta} = -\frac{1}{2V} \sum_i \sum_{j \neq i} r_{ij}^\alpha F_{ij}^\beta + \frac{Nk_B T}{V}. \quad (2.14)$$

Here, α and β can be x , y , and z . In some cases, we are only interested in the diagonal terms:

$$\sigma^{xx} = -\frac{1}{2V} \sum_i \sum_{j \neq i} x_{ij} F_{ij}^x + \frac{Nk_B T}{V}. \quad (2.15)$$

$$\sigma^{yy} = -\frac{1}{2V} \sum_i \sum_{j \neq i} y_{ij} F_{ij}^y + \frac{Nk_B T}{V}. \quad (2.16)$$

¹We use letters in bold face like \mathbf{W} to denote tensors and letters in italic bold face like \mathbf{F} to denote vectors.

$$\sigma^{zz} = -\frac{1}{2V} \sum_i \sum_{j \neq i} z_{ij} F_{ij}^z + \frac{Nk_B T}{V}. \quad (2.17)$$

If the system is isotropic, we usually average these diagonal terms to get a scalar, which is usually called the pressure:

$$p = \frac{1}{3} (\sigma^{xx} + \sigma^{yy} + \sigma^{zz}) = -\frac{1}{6V} \sum_i \sum_{j \neq i} \mathbf{r}_{ij} \cdot \mathbf{F}_{ij} + \frac{Nk_B T}{V}. \quad (2.18)$$

2.4.3 Heat current

GPUMD can be used to compute the lattice thermal conductivity using the Green-Kubo formula, which requires calculating the microscopic heat current. Similar to stress, heat current consists of two parts, a part related to force (usually called the potential part) and a part related to particle movements (usually called the kinetic part or the convective part). The potential part should be calculated along with force evaluation.

In classical physics, the heat current vector \mathbf{J} is defined to be the time derivative of the sum of the energy moments:

$$\mathbf{J} = \frac{d}{dt} \sum_i \mathbf{r}_i E_i. \quad (2.19)$$

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

$$E_i = \frac{1}{2} m_i \mathbf{v}_i^2 + U_i. \quad (2.20)$$

Using Leibniz's rule, we have

$$\mathbf{J} = \sum_i \mathbf{v}_i E_i + \sum_i \mathbf{r}_i \frac{d}{dt} E_i. \quad (2.21)$$

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

$$\mathbf{J}^{\text{pot}} = \sum_i \mathbf{r}_i \frac{dE_i}{dt} \quad (2.22)$$

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 along the transport directions). For two-body potentials, we can arrive at the following expression which is suitable for implementation:

$$\mathbf{J}^{\text{pot}} = -\frac{1}{2} \sum_i \sum_{j \neq i} \mathbf{r}_{ij} (\mathbf{F}_{ij} \cdot \mathbf{v}_i). \quad (2.23)$$

This equation can be expressed in an equivalent way:

$$\mathbf{J}^{\text{pot}} = -\frac{1}{2} \sum_i \sum_{j \neq i} (\mathbf{r}_{ij} \otimes \mathbf{F}_{ij}) \cdot \mathbf{v}_i. \quad (2.24)$$

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

$$\mathbf{J}^{\text{pot}} = \sum_i \mathbf{W}_i \cdot \mathbf{v}_i. \quad (2.25)$$

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

$$\mathbf{J}^{\text{pot}} = \sum_i \mathbf{J}_i^{\text{pot}}, \quad (2.26)$$

$$\mathbf{J}_i^{\text{pot}} = \mathbf{W}_i \cdot \mathbf{v}_i. \quad (2.27)$$

However, we note that the above formula only applies to two-body potentials. For many-body potentials, it has been demonstrated [3] that the above virial-based formula is wrong and the correct one is

$$\mathbf{J}^{\text{pot}} = \sum_i \mathbf{J}_i^{\text{pot}}, \quad (2.28)$$

$$\mathbf{J}_i^{\text{pot}} = \mathbf{r}_{ij} \left(\frac{\partial U_j}{\partial \mathbf{r}_{ji}} \cdot \mathbf{v}_i \right). \quad (2.29)$$

We now have enough ingredients for the force evaluation kernel. Algorithm 7 presents a pseudo code for this.

Algorithm 7 The force evaluation kernel (one thread for one particle).

Require: b is the block index

Require: t is the thread index

Require: S_b is the block size

Require: $i = S_b \times b + t$

Require: N is the number of particles

Require: NN_i is the number neighbours for particle i

Require: NL_{ik} is the index of the k th neighbour of particle i

Require: \mathbf{r}_i is the position vector for particle i

Require: \mathbf{v}_i is the velocity vector for particle i

Require: \mathbf{F}_i is total force on particle i

Require: \mathbf{W}_i is per-atom virial for particle i

Require: \mathbf{J}_i is the per-atom heat current of the potential part for particle i

Require: U_i , \mathbf{F}_i , \mathbf{W}_i , and \mathbf{J}_i are defined using registers or shared memory

1: initialize U_i , \mathbf{F}_i , \mathbf{W}_i , and \mathbf{J}_i to zero

2: **if** $i < N$ **then**

3: read in \mathbf{r}_i to registers from global memory

4: read in \mathbf{v}_i to registers from global memory

5: **for** $k = 0$ to $\text{NN}_i - 1$ **do**

6: $j \leftarrow \text{NL}_{ik}$

7: Read in \mathbf{r}_j from global memory and calculate \mathbf{r}_{ij}

8: apply the minimum image convention to \mathbf{r}_{ij}

9: calculate U_{ij} , $\frac{\partial U_i}{\partial \mathbf{r}_{ij}}$ and $\frac{\partial U_j}{\partial \mathbf{r}_{ji}}$

10: accumulate the potential energy for particle i : $U_i \leftarrow U_i + \frac{1}{2}U_{ij}$

11: accumulate the force on particle i : $\mathbf{F}_i \leftarrow \mathbf{F}_i + \left(\frac{\partial U_i}{\partial \mathbf{r}_{ij}} - \frac{\partial U_j}{\partial \mathbf{r}_{ji}} \right)$

12: accumulate the per-atom virial: $\mathbf{W}_i \leftarrow \mathbf{W}_i - \frac{1}{2}\mathbf{r}_{ij} \otimes \left(\frac{\partial U_i}{\partial \mathbf{r}_{ij}} - \frac{\partial U_j}{\partial \mathbf{r}_{ji}} \right)$

13: accumulate the per-atom heat current: $\mathbf{J}_i \leftarrow \mathbf{J}_i + \mathbf{r}_{ij} \left(\frac{\partial U_j}{\partial \mathbf{r}_{ji}} \cdot \mathbf{v}_i \right)$

14: **end for**

15: save the per-atom quantities U_i , \mathbf{F}_i , \mathbf{W}_i , and \mathbf{J}_i to global memory

16: **end if**

2.5 Integration by one step

The aim of time evolution is to find the phase trajectory

$$\{\mathbf{r}_i(t_1), \mathbf{v}_i(t_1)\}_{i=1}^N, \{\mathbf{r}_i(t_2), \mathbf{v}_i(t_2)\}_{i=1}^N, \dots \quad (2.30)$$

starting from the initial phase point

$$\{\mathbf{r}_i(t_0), \mathbf{v}_i(t_0)\}_{i=1}^N. \quad (2.31)$$

The time interval between two time points $\Delta t = t_1 - t_0 = t_2 - t_1 = \dots$ 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 molecular dynamics 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 (stress) 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.

2.5.1 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 integrator which has the property of time-reversibility is the so-called velocity-Verlet method [5]. 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:

$$\mathbf{v}_i(t_{m+1}) \approx \mathbf{v}_i(t_m) + \frac{\mathbf{F}_i(t_m) + \mathbf{F}_i(t_{m+1})}{2m_i} \Delta t, \quad (2.32)$$

$$\mathbf{x}_i(t_{m+1}) \approx \mathbf{x}_i(t_m) + \mathbf{v}_i(t_m) \Delta t + \frac{1}{2} \frac{\mathbf{F}_i(t_m)}{m_i} (\Delta t)^2, \quad (2.33)$$

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 [15]. In this approach, the time-evolution of a classical system by one step can be formally expressed as

$$\begin{pmatrix} \mathbf{r}_i(t + \Delta t) \\ \mathbf{p}_i(t + \Delta t) \end{pmatrix} = e^{iL\Delta t} \begin{pmatrix} \mathbf{r}_i(t) \\ \mathbf{p}_i(t) \end{pmatrix} \quad (2.34)$$

where \mathbf{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 in quantum mechanics.

The operator iL in the exponent of the evolution operator is called the Liouville operator and is defined by

$$iL(\text{anything}) = \{\text{anything}, H\} \equiv \sum_{i=1}^N \left(\frac{\partial H}{\partial \mathbf{p}_i} \cdot \frac{\partial}{\partial \mathbf{r}_i} - \frac{\partial H}{\partial \mathbf{r}_i} \cdot \frac{\partial}{\partial \mathbf{p}_i} \right) (\text{anything}). \quad (2.35)$$

Here, H is the Hamiltonian of the system. Because

$$\frac{\partial H}{\partial \mathbf{p}_i} = \frac{\mathbf{p}_i}{m_i} \text{ and } -\frac{\partial H}{\partial \mathbf{r}_i} = \mathbf{F}_i, \quad (2.36)$$

we have

$$iL = iL_1 + iL_2, \quad (2.37)$$

$$iL_1 = \sum_{i=1}^N \frac{\mathbf{p}_i}{m_i} \cdot \frac{\partial}{\partial \mathbf{r}_i}, \quad (2.38)$$

$$iL_2 = \sum_{i=1}^N \mathbf{F}_i \cdot \frac{\partial}{\partial \mathbf{p}_i}. \quad (2.39)$$

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 [16], which can be used to derive the following approximation:

$$e^{iL\Delta t} \approx e^{iL_2\Delta t/2}e^{iL_1\Delta t}e^{iL_2\Delta t/2}. \quad (2.40)$$

Now, we can express the one-step integration as

$$\begin{pmatrix} \mathbf{r}_i(t + \Delta t) \\ \mathbf{p}_i(t + \Delta t) \end{pmatrix} \approx e^{iL_2\Delta t/2}e^{iL_1\Delta t}e^{iL_2\Delta t/2} \begin{pmatrix} \mathbf{r}_i(t) \\ \mathbf{p}_i(t) \end{pmatrix}. \quad (2.41)$$

To make further derivations, we note that for an arbitrary constant c and a function $f(x)$, we have $e^{c\frac{\partial}{\partial x}}f(x) = f(x + c)$. Applying this identity to the right most operator in the above equation, we have

$$\begin{pmatrix} \mathbf{r}_i(t + \Delta t) \\ \mathbf{p}_i(t + \Delta t) \end{pmatrix} \approx e^{iL_2\Delta t/2}e^{iL_1\Delta t} \begin{pmatrix} \mathbf{r}_i(t) \\ \mathbf{p}_i(t) + \frac{\Delta t}{2}\mathbf{F}_i(t) \end{pmatrix}. \quad (2.42)$$

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

$$\begin{pmatrix} \mathbf{r}_i(t + \Delta t) \\ \mathbf{p}_i(t + \Delta t) \end{pmatrix} \approx e^{iL_2\Delta t/2} \begin{pmatrix} \mathbf{r}_i(t) + \Delta t \frac{\mathbf{p}_i(t) + \frac{\Delta t}{2}\mathbf{F}_i(t)}{m_i} \\ \mathbf{p}_i(t) + \frac{\Delta t}{2}\mathbf{F}_i(t) \end{pmatrix}. \quad (2.43)$$

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

$$\begin{pmatrix} \mathbf{r}_i(t + \Delta t) \\ \mathbf{p}_i(t + \Delta t) \end{pmatrix} \approx \begin{pmatrix} \mathbf{r}_i(t) + \Delta t \frac{\mathbf{p}_i(t) + \frac{\Delta t}{2}\mathbf{F}_i(t)}{m_i} \\ \mathbf{p}_i(t) + \frac{\Delta t}{2}\mathbf{F}_i(t) + \frac{\Delta t}{2}\mathbf{F}_i(t + \Delta t) \end{pmatrix}. \quad (2.44)$$

It is clear that this equation is equivalent to Eqs. (2.32) and (2.33). The evolution-operator approach is not only more rigorous, but also more handy. As can be seen from the above derivations, one only needs to apply the Liouville operators one after another, which can be easily implemented in a computer language. Because of this, the evolution-operator approach is also called the “direct-translation” approach.

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 8 gives the pseudo code for the complete time-stepping in the NVE ensemble, including force updating. Algorithms 9 and 10 are the pseudo codes for the first and second steps of the velocity-Verlet integrator.

Algorithm 8 The whole time-stepping in the NVE ensemble.

- 1: apply the first step of the velocity-Verlet algorithm
 - 2: update the forces
 - 3: apply the second step of the velocity-Verlet algorithm
-

Algorithm 9 The first step of the velocity-Verlet algorithm.

- Require:** b is the block index
Require: t is th thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$ is the particle index
Require: Δt is the time step
Require: N is the number of particles
Require: m_i is the mass of particle i
Require: \mathbf{r}_i is the position vector of particle i
Require: \mathbf{v}_i is the velocity vector of particle i
Require: \mathbf{F}_i is the total force on particle i obtained within the last time step
- 1: **if** $i < N$ **then**
 - 2: update the velocity partially: $\mathbf{v}_i \leftarrow \mathbf{v}_i + \frac{1}{2} \frac{\mathbf{F}_i}{m_i} \Delta t$
 - 3: update the position fully: $\mathbf{r}_i \leftarrow \mathbf{r}_i + \mathbf{v}_i \Delta t$
 - 4: **end if**
-

2.5.2 Berendsen thermostat and barostat

The Berendsen thermostat and barostat are very suitable for equilibrating the system to a target temperature and pressure.

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 11. For the NPT ensemble, the Berendsen barostat requires an extra scaling of positions and box lengths, as shown in Algorithm 12.

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

$$\mathbf{v}_i^{\text{scaled}} = \mathbf{v}_i \sqrt{1 + \alpha_T \left(\frac{T_0}{T} - 1 \right)}. \quad (2.45)$$

Here, α_T is a dimensionless parameter, T_0 is the target temperature, and T is the instant temperature calculated from the current velocities $\{\mathbf{v}_i\}$. The parameter α_T should be positive

Algorithm 10 The second step of the velocity-Verlet algorithm.

- Require:** b is the block index
Require: t is th thread index
Require: S_b is the block size
Require: $i = S_b \times b + t$ is the particle index
Require: Δt is the time step
Require: N is the number of particles
Require: m_i is the mass of particle i
Require: \mathbf{v}_i is the velocity vector of particle i
Require: \mathbf{F}_i is the total force on particle i obtained within the current time step
- 1: **if** $i < N$ **then**
 - 2: complete the update of the velocity: $\mathbf{v}_i \leftarrow \mathbf{v}_i + \frac{1}{2} \frac{\mathbf{F}_i}{m_i} \Delta t$
 - 3: **end if**
-

Algorithm 11 The whole time-stepping in the NVT ensemble using the Berendsen method.

- 1: apply the first step of velocity-Verlet algorithm
 - 2: update the forces
 - 3: apply the second step of velocity-Verlet algorithm
 - 4: scale the velocities
-

Algorithm 12 The whole time-stepping in the NPT ensemble using the Berendsen method.

- 1: apply the first step of velocity-Verlet algorithm
 - 2: update the forces
 - 3: apply the second step of velocity-Verlet algorithm
 - 4: scale the velocities
 - 5: scale the positions and box lengths
-

and not larger than 1. When $\alpha_T = 1$, the above formula reduces to the simple velocity-scaling formula:

$$\mathbf{v}_i^{\text{scaled}} = \mathbf{v}_i \sqrt{\frac{T_0}{T}}. \quad (2.46)$$

A smaller α_T represents a weaker coupling between the system and the thermostat. Usually, any value of α_T in the range of $0.001 \sim 1$ can be used. Algorithm 13 is the pseudo code for the velocity scaling in the Berendsen thermostat.

Algorithm 13 Velocity scaling in the Berendsen thermostat.

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: \mathbf{v}_i is the velocity vector of particle i

- 1: **if** $i < N$ **then**
 - 2: factor $\leftarrow \sqrt{1 + \alpha_T \left(\frac{T_0}{T} - 1 \right)}$
 - 3: $\mathbf{v}_i \leftarrow \mathbf{v}_i \times \text{factor}$
 - 4: **end if**
-

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

$$\mathbf{r}_i^{\text{scaled}} = \mathbf{r}_i [1 - \alpha_p (\mathbf{p}_0 - \mathbf{p})]. \quad (2.47)$$

Here, α_p is a parameter and \mathbf{p}_0 (\mathbf{p}) is the target (instant) pressure along 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. Algorithm 14 is the pseudo code for the position and box scaling in the Berendsen barostat.

2.5.3 Nosé-Hoover chain thermostat

The Nosé-Hoover chain method [7–9, 11, 15] is more suitable for calculating equilibrium properties in a specific ensemble. In the current version of GPUMD, only the Nosé-Hoover

Algorithm 14 Pseudo code for the position/box scaling in the Berendsen barostat.

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: x_i is the x -coordinate of particle i

Require: L_i is the box length along the x -direction

1: **if** $i < N$ **then**

2: **if** use periodic boundary conditions along the x direction **then**

3: factor $\leftarrow 1 - \alpha_p(p_{0x} - p_x)$

4: scale the positions along the x -direction: $x_i \leftarrow x_i \times \text{factor}$

5: scale the box length along the x -direction: $L_x \leftarrow L_x \times \text{factor}$

6: **end if**

7: Do the same for the other directions

8: **end if**

chain thermostat is implemented. We hope to implement the Nosé-Hoover chain barostat in a future version.

We start with the well-known Nosé-Hoover equations of motion [7–9]:

$$\frac{d}{dt}\mathbf{r}_i = \frac{\mathbf{p}_i}{m_i}, \quad (2.48)$$

$$\frac{d}{dt}\mathbf{p}_i = \mathbf{F}_i - \frac{\pi}{Q}\mathbf{p}_i, \quad (2.49)$$

$$\frac{d}{dt}\eta = \frac{\pi}{Q}, \quad (2.50)$$

$$\frac{d}{dt}\pi = 2 \left(\sum_i \frac{\mathbf{p}_i^2}{2m_i} - dN \frac{k_B T}{2} \right), \quad (2.51)$$

where d is the dimension of the space (which is 3 in most applications), \mathbf{r}_i , \mathbf{p}_i , \mathbf{F}_i , and m_i are the position, momentum, force, and mass of particle i , and η , π , and Q are the “position”, “momentum”, and “mass” of the thermostat. Actually, η is dimensionless, π has the dimension of [energy] \times [time] and Q has the dimension of [energy] \times [time]².

The Nosé-Hoover Hamiltonian can generate canonical ensemble in some cases, but it can fail when more than one conservation law is obeyed by the system. The major reason for this failure is that the degree of freedom of the thermostat itself does not follow the canonical distribution. To overcome this difficulty, Martyna *et al.* [9] proposed the Nosé-Hoover chain method, where a chain of M thermostats are introduced and the temperature of the m th thermostat is controlled by the $(m+1)$ th thermostat. The temperature of the system is controlled by the 1st thermostat. When $M = 1$, the Nosé-Hoover chain method reduces to the Nosé-Hoover method.

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

$$\frac{d}{dt}\mathbf{r}_i = \frac{\mathbf{p}_i}{m_i}, \quad (2.52)$$

$$\frac{d}{dt}\mathbf{p}_i = \mathbf{F}_i - \frac{\pi_0}{Q_0}\mathbf{p}_i, \quad (2.53)$$

$$\frac{d}{dt}\eta_k = \frac{\pi_k}{Q_k} \quad (k = 0, 1, \dots, M-1), \quad (2.54)$$

$$\frac{d}{dt}\pi_0 = 2 \left(\sum_i \frac{\mathbf{p}_i^2}{2m_i} - dN \frac{k_B T}{2} \right) - \frac{\pi_1}{Q_1} \pi_0, \quad (2.55)$$

$$\frac{d}{dt}\pi_k = 2 \left(\frac{\pi_{k-1}^2}{2Q_{k-1}} - \frac{k_B T}{2} \right) - \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \quad (k = 1, 2, \dots, M-2), \quad (2.56)$$

$$\frac{d}{dt}\pi_{M-1} = 2 \left(\frac{\pi_{M-2}^2}{2Q_{M-2}} - \frac{k_B T}{2} \right). \quad (2.57)$$

The optimal choice [9] for the thermostat masses is

$$Q_0 = dNk_B T \tau^2, \quad (2.58)$$

$$Q_k = k_B T \tau^2 \quad (k = 1, 2, \dots, M-1), \quad (2.59)$$

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 NVT ensemble using the Nosé-Hoover chain can also be formulated using the approach of the time-evolution operator [11, 15]. The total Liouville operator for the equations of motion in the Nosé-Hoover chain method is [11, 15]

$$iL = iL_1 + iL_2 + iL_T, \quad (2.60)$$

$$iL_1 = \sum_{i=1}^N \frac{\mathbf{p}_i}{m_i} \cdot \frac{\partial}{\partial \mathbf{r}_i}, \quad (2.61)$$

$$iL_2 = \sum_{i=1}^N \mathbf{F}_i \cdot \frac{\partial}{\partial \mathbf{p}_i}. \quad (2.62)$$

$$iL_T = \sum_{k=0}^{M-1} \frac{\pi_k}{Q_k} \frac{\partial}{\partial \eta_k} + \sum_{k=0}^{M-2} \left(G_k - \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \right) \frac{\partial}{\partial \pi_k} + G_{M-1} \frac{\partial}{\partial \pi_{M-1}} - \sum_{i=0}^{N-1} \frac{\pi_0}{Q_0} \mathbf{p}_i \cdot \frac{\partial}{\partial \mathbf{p}_i}. \quad (2.63)$$

That is, the Liouville operator for the NVT ensemble contains an extra term iL_T related to the thermostats, 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:

$$e^{iL} \approx e^{iL_T \Delta t/2} e^{iL_2 \Delta t/2} e^{iL_1 \Delta t} e^{iL_2 \Delta t/2} e^{iL_T \Delta t/2}. \quad (2.64)$$

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 :

$$iL_T = iL_{T1} + iL_{T2} + iL_{T3}, \quad (2.65)$$

$$iL_{T1} = \sum_{k=0}^{M-1} \frac{\pi_k}{Q_k} \frac{\partial}{\partial \eta_k}, \quad (2.66)$$

$$iL_{T2} = \sum_{k=0}^{M-2} \left(G_k - \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \right) \frac{\partial}{\partial \pi_k} + G_{M-1} \frac{\partial}{\partial \pi_{M-1}}, \quad (2.67)$$

$$iL_{T3} = \sum_{i=0}^{N-1} \frac{\pi_0}{Q_0} \mathbf{p}_i \cdot \frac{\partial}{\partial \mathbf{p}_i}. \quad (2.68)$$

We can then make the following factorization:

$$e^{iL_T \Delta t/2} \approx e^{iL_{T2} \Delta t/4} e^{iL_{T3} \Delta t/2} e^{iL_{T1} \Delta t/2} e^{iL_{T2} \Delta t/4}. \quad (2.69)$$

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

$$e^{iL_{T2} \Delta t/4} \approx \prod_{k=0}^{M-2} \left(e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} e^{\frac{\Delta t}{4} G_k \frac{\partial}{\partial \pi_k}} e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} \right) e^{\frac{\Delta t}{4} G_{M-1} \frac{\partial}{\partial \pi_{M-1}}} \quad (2.70)$$

and correspondingly factorize that on the left as

$$e^{iL_{T2} \Delta t/4} \approx e^{\frac{\Delta t}{4} G_{M-1} \frac{\partial}{\partial \pi_{M-1}}} \prod_{k=M-2}^0 \left(e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} e^{\frac{\Delta t}{4} G_k \frac{\partial}{\partial \pi_k}} e^{-\frac{\Delta t}{8} \frac{\pi_{k+1}}{Q_{k+1}} \pi_k \frac{\partial}{\partial \pi_k}} \right). \quad (2.71)$$

We are now prepared to set up the algorithms for the integrator in the NVT ensemble using the Nosé-Hoover chain method. This can be easily done by the “direct-translation” procedure. Before writing down the algorithms, we discuss a technical point. 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 :

$$e^{cx \frac{\partial}{\partial x}} x = e^c x. \quad (2.72)$$

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 $e^{iL_T \Delta t/2}$, it does not affect the thermostats. So, 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 thermostats and that for the particles are separated.

Algorithm 15 presents the pseudo code for the whole time-stepping in the NVT ensemble using the Nosé-Hoover chain method. In this algorithm, the only lines 1 and 7 do the same thing: update the thermostat variables. This is a very cheap calculation and we only implement it on the CPU. Its implementation is straightforward using the “direct-translation” technique.

Algorithm 15 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: apply the first step of velocity-Verlet algorithm corresponding to the operator $e^{iL_1 \Delta t} e^{iL_2 \Delta t/2}$
 - 4: update the forces
 - 5: apply the second step of velocity-Verlet algorithm corresponding to the operator $e^{iL_2 \Delta t/2}$
 - 6: apply the operator $e^{iL_T \Delta t/2}$ except for $e^{iL_{T3} \Delta t/2}$ within it and save the value of $e^{-(\pi_0/Q_0) \Delta t/2}$
 - 7: scale the velocity components of all the particles by the factor $e^{-(\pi_0/Q_0) \Delta t/2}$
-

2.6 Postprocess

2.6.1 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 [17, 18]. 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):

$$\kappa_{xx}(t) = \frac{1}{k_B T^2 V} \int_0^t dt' \text{HAC}_{xx}(t'). \quad (2.73)$$

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

$$\text{HAC}_{xx}(t) = \langle J_x(0) J_x(t) \rangle, \quad (2.74)$$

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 \mathbf{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 M heat current data, we can calculate at most M HAC data $\text{HAC}_{xx}(t)$, with $t = 0, \Delta\tau, 2\Delta\tau, \dots, (M-1)\Delta\tau$. 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 ($M \times \Delta\tau$) is much larger than the maximum correlation time t_{\max} one needs. The number of HAC data N_c is related to the maximum correlation time by $t_{\max} = (N_c - 1)\Delta\tau$. In most cases, $N_c = M/10$ is a good choice. It is also convenient to use the same number of time origins, $M - N_c$, to do the time-average for each correlation time. With these considerations, we arrive at the following explicit expression for the HAC:

$$\text{HAC}_{xx}(n_c \Delta\tau) = \frac{1}{M - N_c} \sum_{m=0}^{M-N_c-1} J_x(m\Delta\tau) J_x((m+n_c)\Delta\tau), \quad (2.75)$$

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

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. Algorithm 16 presents the pseudo code for the calculation of $\text{HAC}_{xx}(t)$ from the heat current data saved in the global memory.

2.6.2 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; also called vibrational density of states) [19].

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

$$\langle v_{xi}(0) v_{xi}(t) \rangle. \quad (2.76)$$

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:

$$\text{VAC}_{xx}(t) = \frac{1}{N} \sum_{i=1}^N \langle v_{xi}(0) v_{xi}(t) \rangle. \quad (2.77)$$

Algorithm 16 Calculation of $\text{HAC}_{xx}(t)$ from the heat current.

Require: i is the block index**Require:** j is the thread index**Require:** S_b is the block size**Require:** M is the number of heat current data in each direction**Require:** N_c is the number of HAC data in each direction to be calculated**Require:** J_x is the heat current in the x -direction and is in the global memory**Require:** $\text{HAC}_{xx}[i][j]$ is the HAC in the x -direction and is defined in the shared memory. Note that the index i corresponds to the block index and the index j corresponds to the thread index.1: initialize before accumulation: $\text{HAC}_{xx}[i][j] \leftarrow 0$ 2: **for** $n = 0$ to $\lceil (M - N_c)/S_b \rceil - 1$ **do**3: **if** $j + nS_b < M - N_c$ **then**4: $\text{HAC}_{xx}[i][j] \leftarrow \text{HAC}_{xx}[i][j] + J_x[j + nS_b]J_x[j + nS_b + i]$ 5: **end if**6: **end for**

7: synchronize the threads within each block

8: use binary reduction to do the summation in Eq. (2.75): $\text{HAC}_{xx}[i][0] \leftarrow \sum_j \text{HAC}_{xx}[i][j]$ 9: save $\text{HAC}_{xx}[i][0]/(M - N_c)$ to the global memory

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

$$\text{VAC}_{xx}(t) = \left\langle \frac{1}{N} \sum_{i=1}^N v_{xi}(0)v_{xi}(t) \right\rangle. \quad (2.78)$$

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

$$\text{VAC}_{xx}(n_c \Delta\tau) = \frac{1}{(M - N_c)N} \sum_{m=0}^{M-N_c-1} \sum_{i=1}^N v_{xi}(m\Delta\tau)v_{xi}((m + n_c)\Delta\tau), \quad (2.79)$$

where $n_c = 0, 1, 2, \dots, N_c - 1$. 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 $D_{xx}(t)$ by

$$D_{xx}(t) = \int_0^t dt' \text{VAC}_{xx}(t'). \quad (2.80)$$

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

$$D_{xx}(t) = \frac{1}{2} \frac{d}{dt} \Delta x^2(t), \quad (2.81)$$

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

$$\Delta x^2(t) = \left\langle \frac{1}{N} \sum_{i=1}^N [x_i(t) - x_i(0)]^2 \right\rangle = \frac{1}{N} \sum_{i=1}^N \langle [x_i(t) - x_i(0)]^2 \rangle. \quad (2.82)$$

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

$$x_i(t) - x_i(0) = \int_0^t dt' v_{xi}(t'), \quad (2.83)$$

we have

$$[x_i(t) - x_i(0)]^2 = \int_0^t dt' v_{xi}(t') \int_0^t dt'' v_{xi}(t'') = \int_0^t dt' \int_0^t dt'' v_{xi}(t') v_{xi}(t''). \quad (2.84)$$

Then, the MSD can be expressed as

$$\Delta x^2(t) = \frac{1}{N} \sum_{i=1}^N \int_0^t dt' \int_0^t dt'' \langle v_{xi}(t') v_{xi}(t'') \rangle. \quad (2.85)$$

Using Leibniz's rule, we have

$$D_{xx}(t) = \frac{1}{2} \frac{d}{dt} \Delta x^2(t) = \frac{1}{N} \sum_{i=1}^N \int_0^t dt' \langle v_{xi}(t) v_{xi}(t') \rangle, \quad (2.86)$$

which can be rewritten as

$$D_{xx}(t) = \frac{1}{N} \sum_{i=1}^N \int_0^t dt' \langle v_{xi}(0) v_{xi}(t' - t) \rangle. \quad (2.87)$$

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

$$D_{xx}(t) = \frac{1}{N} \sum_{i=1}^N \int_{-t}^0 d\tau \langle v_{xi}(0) v_{xi}(\tau) \rangle, \quad (2.88)$$

which can be rewritten as

$$D_{xx}(t) = \frac{1}{N} \sum_{i=1}^N \int_{-t}^0 d\tau \langle v_{xi}(-\tau) v_{xi}(0) \rangle. \quad (2.89)$$

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

$$D_{xx}(t) = \frac{1}{N} \sum_{i=1}^N \int_0^t dt' \langle v_{xi}(t') v_{xi}(0) \rangle = \int_0^t dt' \text{VAC}_{xx}(t). \quad (2.90)$$

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

In summary,

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

It is interesting that the same VAC can be used to compute the PDOS, as first demonstrated by Dickey and Paskin [19]. The PDOS is simply the Fourier transform of the normalized VAC:

$$\rho_x(\omega) = \int_{-\infty}^{\infty} dt e^{i\omega t} \text{VAC}_{xx}(t). \quad (2.91)$$

Here, $\text{VAC}_{xx}(t)$ should be understood at the normalized function $\text{VAC}_{xx}(t)/\text{VAC}_{xx}(0)$. Although it looks simple, it does not mean that you 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, $\text{VAC}_{xx}(-t) = \text{VAC}_{xx}(t)$. Using this, we have

$$\rho_x(\omega) = \int_{-\infty}^{\infty} dt \cos(\omega t) \text{VAC}_{xx}(t). \quad (2.92)$$

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:

$$\rho_x(\omega) \approx \sum_{n_c=0}^{N_c-1} (2 - \delta_{n_c0}) \Delta\tau \cos(\omega n_c \Delta\tau) \text{VAC}_{xx}(n_c \Delta\tau). \quad (2.93)$$

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:

$$\rho_x(\omega) \approx \sum_{n_c=0}^{N_c-1} (2 - \delta_{n_c0}) \Delta\tau \cos(\omega n_c \Delta\tau) \text{VAC}_{xx}(n_c \Delta\tau) H(n_c), \quad (2.94)$$

$$H(n_c) = \frac{1}{2} \left[\cos\left(\frac{\pi n_c}{N_c}\right) + 1 \right]. \quad (2.95)$$

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:

$$\text{VAC}_{xx}(t) = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi} e^{-i\omega t} \rho_x(\omega). \quad (2.96)$$

As we have normalized the VAC, we have

$$1 = \text{VAC}_{xx}(0) = \int_{-\infty}^{\infty} \frac{d\omega}{2\pi} \rho_x(\omega). \quad (2.97)$$

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

$$\int_0^{\infty} \frac{d\omega}{2\pi} \rho_x(\omega) = \frac{1}{2}. \quad (2.98)$$

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

Chapter 3

How to use GPUMD

3.1 Compile GPUMD

Go to the `src` directory and type `make` (or `make -j8` to make it faster). You may want to first do `make clean`. The default option is to use GPU. It can be modified to use CPU only.

I should make the `makefile` more user-friendly. Mikko, I need your help!

3.2 Run simulations with GPUMD

To run a simulation or a set of simulations with GPUMD, you need to first prepare a few input files.

3.2.1 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
```

In the first line, `N` is the number of atoms, `M` is the maximum possible number of neighbour atoms for one atom, and `cutoff` is the cutoff distance used for building the neighbour 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 m th atom. This file should have $N + 2$ lines.

In the current version, types and group labels are not used, and they can be set to 0. We will make use of these input data in a future version.

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).

3.2.2 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
potential
velocity

# Then, write (all or part of) these keywords in any order
ensemble
time_step
dump_thermo
dump_position
dump_velocity
dump_force
compute_vac
compute_hac

# Then the keyword run
run

# Now you can repeat the last two groups as many times as you want
-----
```

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. For example, the command

```
potential potentials/graphene.tersoff
```

means that you will use the potential model defined in the file `graphene.tersoff` which should be in the `potentials` folder. We will talk more about the potential files in the next subsection.

2. The `velocity` keyword.

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

```
velocity 10.0
```

means that you want to set the initial temperature to 10 K.

3. The `ensemble` keyword.

This keyword specifies the statistical ensemble and the relevant parameters. The number of parameters depends on the first parameter, which is the ensemble type. There are currently 3 ensemble types: *NVE*, *NVT*, and *NPT*. If the ensemble type is *NVE*, there is no need to further specify any other parameters. Therefore, the command to specify the *NVE* ensemble is:

```
ensemble nve
```

If the ensemble type is *NVT*, you need to specify a target temperature and a parameter which reflects the strength of the coupling between the system and the thermostat. There are two *NVT* methods, the Berendsen method (`nvt_ber`) and the Nosé-Hoover chain method (`nvt_nhc`). The complete command can be

```
ensemble nvt_ber T T_coup
```

or

```
ensemble nvt_nhc T T_coup
```

Here, `T` is the target temperature and `T_coup` is the coupling constant in each method. If the ensemble type is *NPT*, you need to specify a target temperature, a temperature coupling constant, 3 target pressures (along the 3 directions), and a pressure coupling constant. There is currently only one *NPT* method, which is the Berendsen method. The complete command is

```
ensemble nvt_ber T T_coup Px Py Pz P_coup
```

Here, `T` is the target temperature, `T_coup` is the temperature coupling constant in the Berendsen method, `Px`, `Py`, and `Pz` are the target pressure (stress) along the 3 directions, and `P_coup` is the pressure coupling constant in the Berendsen method. The units of temperature and pressure for this keyword are K and GPa (10^9 Pa), 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 [0.001, 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 [10, 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 1.0
```

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 `dump_thermo`, `dump_position`, `dump_velocity`, and `dump_force` keywords.

These keywords only requires a single parameter, which is the output frequency for the relevant quantities: common thermodynamic quantities for `dump_thermo`, positions for `dump_position`, velocities for `dump_velocity`, and forces for `dump_force`. For example, the command

```
dump_thermo 1000
```

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

6. The `compute_vac` keyword.

This keyword is related to the calculations of VAC (velocity autocorrelation) and two other related quantities: RDC (running diffusion constant) and DOS (phonon density of states). If this keyword appears in your input file, it means that you want to calculate the VAC in a run; otherwise, it means that you don't want to calculate the VAC. The first parameter for this keyword is the sample 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 calculation. 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 correlation time is $(200 - 1) \times (5\Delta t)$, where Δt is the time step; (4) the maximum angular frequency you want to consider is $\omega_{\max} = 2\pi\nu_{\max} = 350$ THz. The results will be written to a file named `vac.out` in the same folder where you put your `run.in` file in.

7. 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 sample 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 10 100000 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 10 steps; (3) the maximum correlation time is $(100000 - 1) \times (10\Delta t)$, where Δt is the time step; (4) the HAC/RTC data will be averaged for every 10 data and the number of HAC/RTC data in a given direction is then $100000/10 = 10000$. The results will be written to a file named `hac.out` in the same folder where you put your `run.in` file in.

8. 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 collect 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 ten 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_{run} and the sampling interval as N_{samp} , the memory to be used for holding the velocity data is $(N_{\text{run}}/N_{\text{samp}}) \times N \times 3 \times 8$ byte if using double-precision. If the number of atoms is $N = 10^4$, $N_{\text{samp}} = 5$, and $N_{\text{run}} = 10^5$, the memory to be used for holding the velocity data is 4.8 GB. This is ok for Tesla K40 and K80, but may be too much for older GPUs. The current version of GPUMD is thus not quite suitable to study problems with very large velocity-velocity correlation times. In this case, you can save the velocities into disk (using the `dump_velocity` keyword) and analyse the data by yourself.

3.2.3 Prepare a potential file

The `potential` keyword used in the `run.in` file is used to specify the file which contains the potential model and parameters to be used in a simulation.

We require the following format in a potential file:

```
potential_name
number
parameter_1 parameter_2 ...
```

Here, `potential_name` is the name of the potential model, which can only be `tersoff` or `sw` in the current version. In the second line, `number` should be 1 in the current version. It means that only one atom type is considered. The next line contains the values of the parameters used in the potential model in a given order. Referring to the Ref. [1], the parameters for the Tersoff potential should be given in the following order (units are given in the parentheses):

$$A(\text{eV}) \ B(\text{eV}) \ \lambda(\text{\AA}^{-1}) \ \mu(\text{\AA}^{-1}) \ \beta \ n \ c \ d \ h \ R(\text{\AA}) \ S(\text{\AA}) \quad (3.1)$$

Referring to Ref. [2], the parameters for the Stillinger-Weber potential should be given in the following order (units are given in the parentheses):

$$\epsilon(\text{eV}) \ \lambda \ A \ B \ a \ \gamma \ \sigma(\text{\AA}) \ \cos(\theta_t) \quad (3.2)$$

We have already prepared 3 potential files in the `potentials` folder: `graphene.tersoff`, `silicon.tersoff`, and `silicon.sw`.

We are still working on extending the code to consider more than one atom types.

3.2.4 Prepare a “driver input file”

In the above, we have described how to prepare the individual input files. GPUMD requires you put the `xyz.in` file and the `run.in` file into the same folder. Then, you just need 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 has the following content:

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

This tells the code that there will be one simulation and the input files are prepared in the folder `examples/lattice_constant/si_tersoff`. You can also run multiple simulations in a single round by specifying more input folders in the “driver input file”. An example is:

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

In this case, it means that you want to run 2 simulations consecutively.

Output files will be created in the folders containing the corresponding input files. Note that results will be appended to existing files rather than erasing existing data.

3.2.5 Run the code

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

```
src/gpumd < input
```

3.3 Data formats in the output files

To analyse the results obtained by using GPUMD, you have to know how the output data are organized. We note that for all the output files, results from a new simulation will be appended, rather than erasing existing data. Therefore, if you do not intend to append new results to existing output files, you’d better first delete the existing output files or make a copy of them.

3.3.1 The thermo.out 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) along the x direction
- column 5: pressure (in unit of GPa) along the y direction
- column 6: pressure (in unit of GPa) along the z direction
- column 7: box length (in unit of Å) along the x direction
- column 8: box length (in unit of Å) along the y direction
- column 9: box length (in unit of Å) along the z direction

3.3.2 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 in xyz.out 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)
```

The box lengths at different time points can be found from the thermo.out file, but you have to make sure that you have required to dump data to that file with an appropriate sampling interval.

3.3.3 The v.out file

Similar to the xyz.out file, but for the velocities.

3.3.4 The f.out file

Similar to the xyz.out file, but for the forces.

3.3.5 The vac.out file

This file contains the data of VAC (velocity autocorrelation) and related quantities, namely, the RDC (running diffusion coefficient) and the DOS (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 $\text{\AA}^2/\text{ps}^2$) along the x direction
- column 3: VAC (in unit of $\text{\AA}^2/\text{ps}^2$) along the y direction
- column 4: VAC (in unit of $\text{\AA}^2/\text{ps}^2$) along the z direction
- column 5: RDC (in unit of $\text{\AA}^2/\text{ps}$) along the x direction
- column 6: RDC (in unit of $\text{\AA}^2/\text{ps}$) along the y direction
- column 7: RDC (in unit of $\text{\AA}^2/\text{ps}$) along the z direction
- column 8: angular frequency ω in unit of THz
- column 9: DOS (in unit of $1/\text{THz}$) along the x direction
- column 10: DOS (in unit of $1/\text{THz}$) along the y direction
- column 11: DOS (in unit of $1/\text{THz}$) along the z direction

3.3.6 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: HAC (in unit of eV^3/amu) along the x direction
- column 3: HAC (in unit of eV^3/amu) along the y direction
- column 4: HAC (in unit of eV^3/amu) along the z direction
- column 5: RTC (in unit of $\text{Wm}^{-1}\text{K}^{-1}$) along the x direction
- column 6: RTC (in unit of $\text{Wm}^{-1}\text{K}^{-1}$) along the y direction
- column 7: RTC (in unit of $\text{Wm}^{-1}\text{K}^{-1}$) along the z direction

Chapter 4

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 GPU code.

4.1 Lattice constant of silicon

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 $8 \times 8 \times 8 = 4096$ silicon atoms and compare the results obtained with two different potential models, namely, the Tersoff potential and the Stillinger-Weber potential.

The “run.in” input file in the case of the Tersoff potential is given below. The first line of command tells that the potential to be used is specified in the file `potentials/si.tersoff`. In the case of the Stillinger-Weber potential, this file should be changed to `potentials/si.sw`. The second line of the command tells that the velocities will be initialized with a temperature of 200 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 200 K and the pressures are zero along all the directions. The coupling constants are 0.01 and 0.0005 for the thermostat and the barostat, respectively. The time step for integration is 1 fs. There are 100 000 steps for this run and the thermodynamic quantities will be output every 100 steps. After this run, there are 6 other runs with the same parameters but the temperature. The temperature increases from 200 K to 1 400 K from the first to the last run.

```
-----
potential potentials/si.tersoff # use the Tersoff potential
velocity 200.0

ensemble npt_ber 200.0 0.01 0.0 0.0 0.0 0.0005
time_step 1.0
dump_thermo 100
run 100000

ensemble npt_ber 400.0 0.01 0.0 0.0 0.0 0.0005
dump_thermo 100
run 100000

ensemble npt_ber 600.0 0.01 0.0 0.0 0.0 0.0005
dump_thermo 100
```

```

run          100000

ensemble     npt_ber 800.0 0.01 0.0 0.0 0.0 0.0005
dump_thermo  100
run          100000

ensemble     npt_ber 1000.0 0.01 0.0 0.0 0.0 0.0005
dump_thermo  100
run          100000

ensemble     npt_ber 1200.0 0.01 0.0 0.0 0.0 0.0005
dump_thermo  100
run          100000

ensemble     npt_ber 1400.0 0.01 0.0 0.0 0.0 0.0005
dump_thermo  100
run          100000
-----

```

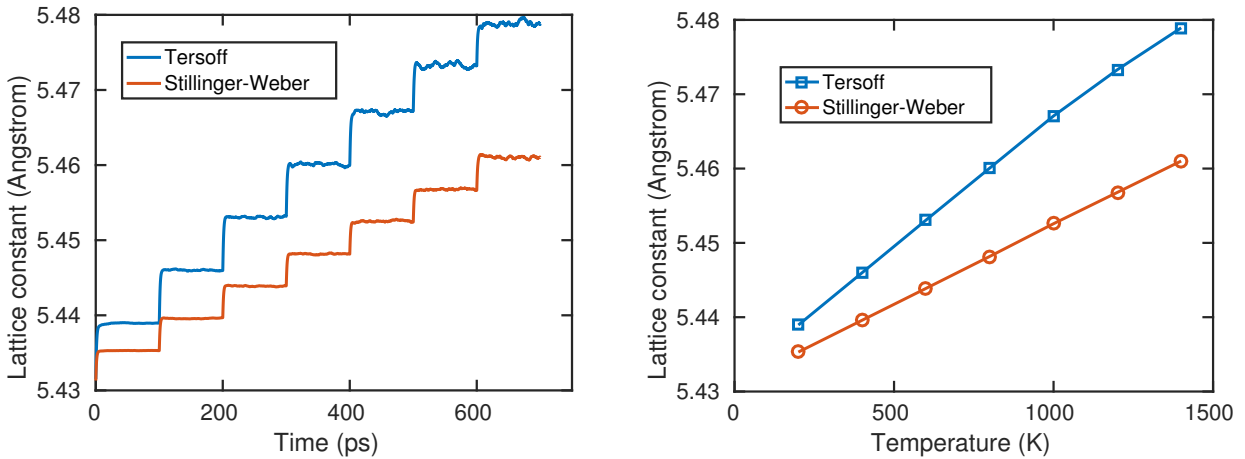


Figure 4.1: (Left) Instant lattice constant of silicon (of the diamond crystal structure) as a function of simulations time. (Right) Average lattice constant (over the last 50 ps in each run for a given temperature) as a function of temperature.

The two simulations take a few minutes in total using a Tesla K40 card. In the output file `thermo.out`, we can get the data for the box lengths (columns 7-9), which can be used to calculate the lattice constant. Because the system is isotropic, the results in different directions are averaged. We have 7 runs, each with 100 ps, and we use the data within the last 50 ps in each run to calculate the average lattice constant at each temperature. The results are shown in Fig. 4.1. The lattice constants obtained by the Tersoff potential are larger than those obtained by the Stillinger-Weber potential, in agreement with the results obtained by Howell [20] using LAMMPS.

4.2 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 “run.in” file

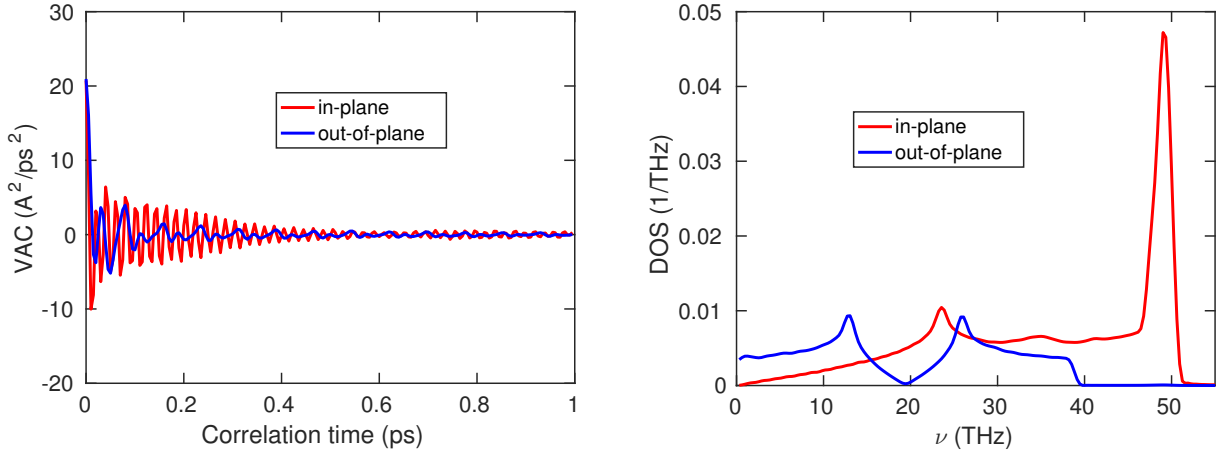


Figure 4.2: Velocity autocorrelation function (left) and phonon density of states (right) for graphene at 300 K and zero pressure. Note that the results are decomposed into an in-plane part and an out-of-plane part, which is special for two-dimensional materials.

reads:

```
-----
potential    potentials/graphene.tersoff
velocity     300.0

ensemble     npt_ber 300 0.01 0.0 0.0 0.0 0.0005
time_step    1.0
dump_thermo  100
run          200000

ensemble     nve
compute_vac  5 200 400.0
run          100000
-----
```

The first two lines specify the potential file `potentials/graphene.tersoff` and the initial temperature, $T = 300$ K. The following 4 non-empty lines constitute the equilibration stage, where the *NPT* ensemble is used. This run lasts 200 ps. The last 3 non-empty lines correspond to the production stage, where 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. This production run lasts 100 ps.

Figure 4.2 shows the calculated VAC and PDOS. For three-dimensional systems, the results along different directions are usually averaged, but for two-dimensional materials like graphene, it is natural to consider the in-plane part (corresponds to the x and y directions in the simulation) and the out-of-plane part (corresponds to the z direction in the simulation) separately. It can be seen that the two components behave very differently.

4.3 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 “run.in” file for this simulation reads:

```

-----
potential          potentials/graphene.tersoff
velocity           10.0

ensemble           npt_ber 300.0 0.01 0.0 0.0 0.0 0.0001
time_step           1.0
dump_thermo        100
run                1000000

ensemble           nve
compute_hac         10 100000 10
run                10000000
-----

```

Most of the commands have been explained in previous examples. The only new command is the line with `compute_hac`. The first number (10) in this line means that the heat current data will be recorded every 10 steps (10 fs). The second number (100000) means that 100 000 HAC data will be calculated along each direction so that the maximum correlation time will be about 1 ns. The last number (10) means that the HAC and RTC data will be averaged for every 10 data points such that the time interval for the output data will be 0.1 ps. The equilibration (using the *NPT* ensemble) stage lasts 1 ns (10^6 steps) and the production (using the *NVE* ensemble) stages last 10 ns (10^7 steps).

Calculating thermal conductivity of graphene can be very time consuming. By running the above simulations tens of times, you may be able to obtain comparable results as presented in Fig. 4(e) in Ref. [3]. We do not present and discuss the results here. An interested reader is encouraged to do the simulations and compare his/her results with those in Ref. [3].

Chapter 5

Publications that are related to GPUMD

GPUMD has found a few successful applications. Here, we collect the relevant publications:

1. Ref. [21]

This paper provides some proof-of-concept thermal conductivity calculations on the GPU.

2. Ref. [3]

In this paper, a set of new formulas for force, virial stress, and heat current were derived in detail. These new formulas are crucial for the implementation of GPUMD. This paper also provides systematic applications of GPUMD to the calculation of thermal conductivities of covalently bonded materials, from three-dimensional silicon and diamond to two-dimensional graphene and quasi-one-dimensional carbon nanotube.

3. Ref. [4]

This manuscript discusses some details of the implementation of GPUMD and its performance.

4. Ref. [22]

In this paper, GPUMD was used to relax large-scale polycrystalline samples and evaluate the grain boundary energies in the samples.

5. Ref. [23]

In this paper, GPUMD was used to predict the lattice thermal conductivity of amorphous graphene at different temperatures.

6. Ref. [24]

In this manuscript, GPUMD has been used to obtain relaxed configurations in large-scale graphene patches, which can then be used to determine the disorder experienced by the charge carriers.

Bibliography

- [1] J. Tersoff, “Modeling solid-state chemistry: Interatomic potentials for multicomponent systems”, Phys. Rev. B **39**, 5566 (1989).
- [2] F. H. Stillinger and T. A. Weber, “Computer simulation of local order in condensed phases of silicon”, Phys Rev B, **31**, 5262 (1985).
- [3] Z. Fan, L. F. C. Pereira, H.-Q. Wang, J.-C. Zheng, D. Donadio, and A. 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 (2015).
- [4] Z. Fan, W. Chen, V. Vierimaa, and A. Harju, “Efficient molecular dynamics simulations with many-body potentials on graphics processing units”, arXiv:1610.03343v1 [physics.comp-ph].
- [5] W. C. Swope, H. C. Andersen, P. H. Berens and K. 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”, J. Chem. Phys. **76**, 637 (1982).
- [6] 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”, J. Chem. Phys. **81**, 3684 (1984).
- [7] S. Nosé, “A unified formulation of the constant temperature molecular-dynamics methods”, J. Chem. Phys. **81**, 511 (1984).
- [8] W. G. Hoover, “Canonical dynamics: Equilibrium phase-space distributions”, Phys. Rev. A **31**, 1695 (1985).
- [9] G. J. Martyna, M. E. Tuckerman and M. L. Klein, “Nose-Hoover chains: the canonical ensemble via continuous dynamics”, J. Chem. Phys. **97**, 2635 (1992).
- [10] G. J. Martyna, D. J. Tobias and M. L. Klein, “Constant pressure molecular dynamics algorithms”, J. Chem. Phys. **101**, 4177 (1994).
- [11] 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).
- [12] M. E. Tuckerman, J. Alejandre, R. López-Rendón, A. L. Jochim and G. J. Martyna, “A Liouville-operator derived measure-preserving integrator for molecular dynamics simulations in the isothermal-isobaric ensemble”, J. Phys. A: Math. Gen. **39** 5629 (2006).
- [13] M. 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 (1984).
- [14] <http://docs.nvidia.com/cuda/cuda-c-programming-guide/>

- [15] Mark E. Tuckerman, “Statistical Mechanics: Theory and Molecular Simulation”, (Oxford University Press, 2010).
- [16] H. F. Trotter, “On the product of semi-groups of operators”, *Proc. Amer. Math. Soc.* **10**, 545 (1959).
- [17] M. S. Green, “Markoff random processes and the statistical mechanics of time-dependent phenomena. II. irreversible processes in fluids”, *J. Chem. Phys.* **22**, 398 (1954).
- [18] R. Kubo, “Statistical-mechanical theory of irreversible processes. I. general theory and simple applications to magnetic and conduction problems”, *J. Phys. Soc. Jpn.* **12**, 570 (1957).
- [19] J. M. Dickey and A. Paskin, “Computer simulation of the lattice dynamics of solids”, *Phys. Rev.* **188**, 1407 (1969).
- [20] P. C. Howell, “Comparison of molecular dynamics methods and interatomic potentials for calculating the thermal conductivity of silicon”, *J. Chem. Phys.* **137**, 224111 (2012).
- [21] Z. Fan, T. Siro, and A. Harju, “Accelerated molecular dynamics force evaluation on graphics processing units for thermal conductivity calculations”, *Comput. Phys. Commun.* **184**, 1414 (2013).
- [22] P. Hirvonen, M. M. Ervasti, Z. Fan, M. Jalalvand, M. Seymour, S. M. V. Allaei, N. Provatas, A. Harju, K. R. Elder, and T. Ala-Nissila, “Multiscale modeling of polycrystalline graphene: A comparison of structure and defect energies of realistic samples from phase field crystal models”, *Phys. Rev. B* **94**, 035414 (2016).
- [23] B. Mortazavi, Z. Fan, L. F. C. Pereira, A. Harju, and T. Rabczuk, “Amorphized graphene: A stiff material with low thermal conductivity”, *Carbon* **103**, 318 (2016).
- [24] Z. Fan, A. Uppstu, and A. Harju, “Dominant source of disorder in graphene: Charged impurities or ripples?”, *arXiv:1605.03715v1 [cond-mat.mes-hall]*.