Pro gradu -thesis
Department of Physics
Master's Programme in Materials Research

PERFORMANCE BENEFITS OF COLLECTIVE VARIABLE-DRIVEN HYPERDYNAMICS METHOD ON THE SIMULATION OF DIFFUSION

Mika Kurki

16.9.2020

Supervisor: Antti Kuronen, University Lecturer

Approver: Kai Nordlund, professor

University of Helsinki
Faculty of Science

PL 64 (Gustaf Hällströmin katu 2)
00014 Helsingin yliopisto

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE

Tiedekunta – Fakultet – Faculty	Koulutusohjelma – Utbildningsprogram – Degree programme
Faculty of Science	Master's Programme in Materials Research

Tekijä – Författare – Author

Mika Kurki

Työn nimi – Arbetets titel – Title

Performance benefits of collective variable-driven hyperdynamics method on the simulation of diffusion

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro Gradu -thesis	September 2020	76

Tiivistelmä – Referat – Abstract

The purpose of this study is to analyze performance benefits of Collective Variable-driven Hyperdynamics (CVHD) method over standard molecular dynamics (MD) simulation. Theory of CVHD is an implementation of the hyperdynamics method with some beneficial features of metadynamics. The original implementation of CVHD was modified to work as an addon for COLVARS package of the LAMMPS simulation software with current software versions. About 70 simulations were run and analyzed to verify functionality of CVHD and compare results with CVHD to those without CVHD. The simulated system was the adatom self-diffusion on Copper 001 surface.

It was found out that CVHD provides a significant performance boost (several order of magnitudes) over standard MD while preserving physical accuracy for simulation of the diffusion, but only in limited temperature range. With high temperatures CVHD doesn't speed up the simulation at all compared to standard MD. With low temperatures, it is possible to achieve statistically meaningful number of diffusion events in temperatures where the same with standard MD would require unreasonable long simulations. But also, CVHD slows down at low enough temperatures so that it is impractical.

It was also found out that the collective variable used in this context is suitable for counting number of adatom diffusion events, which helps analysis of adatom trajectories.

It would be interesting to investigate CVHD more by trying different parametrization and by applying it to also other phenomenon than surface diffusion. The code of CVHD provides possibilities for performance optimizations, for example by utilizing parallel computation.

Avainsanat – Nyckelord – Keywords

Accelerated molecular dynamics, COLVARS, CVHD, Hyperdynamics, LAMMPS, Metadynamics, MD

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Table of Contents

1 Introduction

The purpose of this thesis is to implement an accelerated MD (Molecular Dynamics) method called CVHD (Collective variable-driven hyperdynamics) and analyze the suitability of this method for accelerating simulation of the surface diffusion on copper surface. The CVHD method is first introduced in an article "Merging Metadynamics into Hyperdynamics: Accelerated Molecular Simulations Reaching Time Scales from Microseconds to Seconds" by Kristof M. Bal and Erik C. Neyts at 2015 (Bal & Neyts 2015a). CVHD is an implementation of the hyperdynamics method with some beneficial features of metadynamics. The CVHD method works by shortening waiting time of infrequent events by adding a bias potential energy to the energy minima in the system. The CVHD method resets the biasing potential after every event, this follows hyperdynamics methodology but is just opposite to what is used in metadynamics. CVHD belongs to a family of accelerated MD methods. CVHD consist of two parts; CV is a measure for the state of the system and HD is implementation of hyperdynamics force intended to accelerate simulations.

The timeframe of the diffusion can be long compared to the timeframe of classical MD and thus a lot of computer time might be needed when doing the simulation of the diffusion. Therefore, accelerating the simulation of the diffusion could make the research more efficient by shortening computer time needed for the simulations.

The CVHD implementation used in this thesis is based on a code from Bal & Neyts (Bal & Neyts 2015b). The basis of the implementation is COLVARS (COLLective VARiables) package (Fiorin & Klein & Hénin 2013) of the LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) simulation software (Sandia National Laboratories). CVHD is an extension to the COLVARS package.

Both acceleration and accuracy of the CVHD method were analyzed. Acceleration achieved by CVHD was measured by comparing the number of adatom diffusion events on the surface of the copper during the simulation runs of the same length with and without CVHD. The accuracy of CVHD was measured by comparing activation energy E_A and pre-exponential factor Γ_0 obtained from CVHD simulations to those available in literature for the adatom surface diffusion process.

The purpose of this thesis is not to obtain new physical results but to achieve known simulation results faster using the CVHD method. The CVHD implementation has several parameters for fine-

tuning the algorithm. In this thesis the effect of different parameter combinations for the acceleration of the simulation was not analyzed. According to the work of Bal & Neyts (Bal & Neyts 2015a) the difference in acceleration between different parametrizations of CVHD is minimal compared to acceleration that CVHD provides in general. In this thesis, CVHD was only applied to the simulation of adatom diffusion process on copper surface, however by following the guidelines presented, it is easy to apply CVHD also to the simulation of other processes.

The background of diffusion phenomena and simulation of it is given in chapter 2. After that the theory of CVHD method is explained in chapter 3. Chapter 4 describes the computational implementation of CVHD and its components. Setup of simulations and tools used to run and analyze the simulations are described in chapter 5. Results obtained are in chapter 6. It is pointed out that with CVHD it is possible to achieve significant acceleration in limited temperature range while preserving physical accuracy. Chapters 7 and 8 are conclusion and summary of this study. The source code of CVHD implementation as well as various auxiliary scripts and results referenced in this study are stored in the version management system of University of Helsinki, https://version.helsinki.fi/kurkmika/cvhd-for-lammps.

Computer runs for this thesis were done using University of Helsinki's computing facilities in Finnish Grid and Cloud Infrastructure, persistent identifier urn:nbn:fi:research-infras-2016072533 (CSC). The clusters used were Kale, Ukko2 and Alcyone. During the course of this thesis the file system of Kale was changed to Lustre which caused a partial rewrite of scripts used to run the simulations.

2 Background

2.1 Diffusion

The diffusion is a phenomenon of material transport by atomic motion (Callister & Rethwisch 2014). One typical example of the diffusion is putting two plates of metal (for example copper and nickel) in contact with each other. Then some of copper atoms will move to nickel plate. The concentration of copper on nickel is not constant, concentration is higher the closer to the contact surface we are. In fluid an example of the diffusion is an ink drop in the water. The ink will spread gradually to the larger volume in the water and the concentration of the ink will smoothen. The

diffusion can also happen from gas to solid. For example, it is possible to harden surfaces of steel particles by putting steel in contact with carbon-rich gas. Then carbon atoms will diffuse between surface atoms of the steel, resulting in a more wear-resistant material. It is also possible to think that temperature diffuses from warmer location to colder location. One special case of diffusion is self-diffusion where diffusing atoms are the same species as the material where the diffusion happens. In solid metals self-diffusion means that the atoms in the lattice are changing places with each other due to internal kinetic energy.

From an atomic (and molecular) perspective, diffusion is a stepwise migration of atoms from one place to another. This is what happens in examples above. In solid, atoms are located in lattice sites, and thus in solid the diffusion means that an atom moves from one lattice site to another. For an atom to move to another lattice site, there must be an empty adjacent site and sufficient energy to break bonds with current neighbor atoms. The energy needed to break bonds is the kinetic energy of the atom and thus related to thermal vibration. At a specific temperature, some small fraction of the total number of atoms have enough kinetic energy for diffusive motion. This fraction increases with rising temperature. Thermal vibration energy of an atom varies in time, the average kinetic energy of the atom is related to the temperature of the system, but at an arbitrary moment of time the kinetic energy of the atom can be higher or lower than the average. The longer the time, the greater is the probability that some atom has enough kinetic energy to break bonds with its current neighbors. Thus, diffusion is dependent both from temperature and time. (Callister & Rethwisch 2014).

The diffusion can be viewed also from the point of transition state theory, TST. In TST there is a potential energy surface that consists of potential energy minima (basins) and barriers or saddle points between them (Voter 1997). Also, the term dividing surface can be used for the saddle points. Potential energy minima are states and an atom can move from one state to another by crossing the saddle point and landing to another minima. In the diffusion in solid states potential energy minima are locations in lattice points. TST is applicable for infrequent events, the time between successive events must be long enough that after each transition a Boltzmann distribution of energies can be reached. Diffusion is typically such a slow process that this requirement of TST is fulfilled. TST view of diffusion can be visualized using one-dimensional potential.

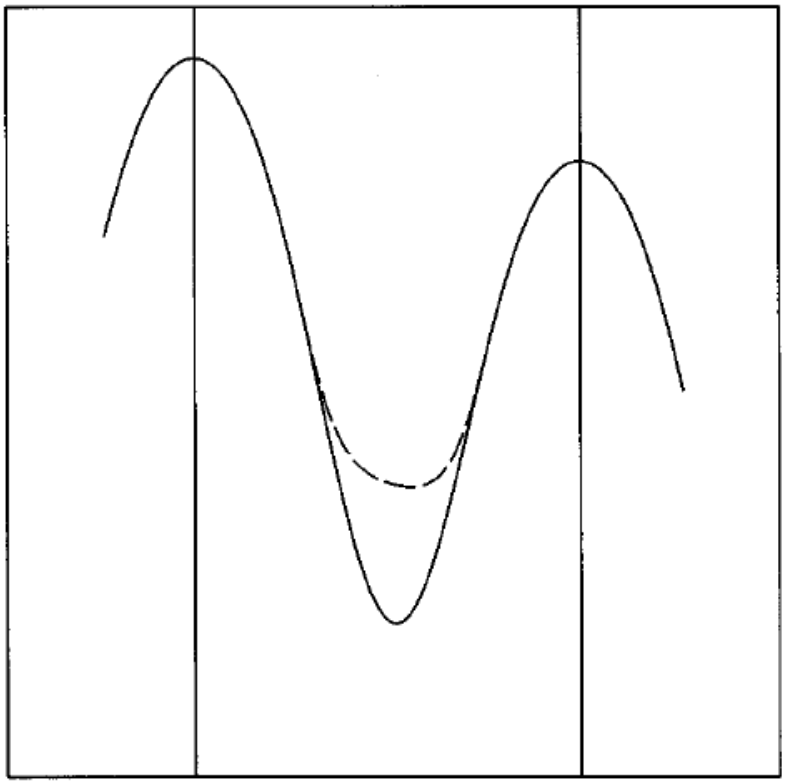

Figure 1:Illustration of one-dimensional potential with one minimum (basin) in the middle and saddle points separating the middle minimum from two other minima in right and left. Saddle points are marked with vertical lines. In this illustration a diffusion event would be the movement of an atom from the minimum in the middle to another minimum in left or right over the saddle point. The dashed line is a bias potential that acceleration methods, like CVHD, produce in order to lower the barrier between minima. The illustration is from article "A method for accelerating the molecular dynamics simulation of infrequent events" (Voter 1997).

The basic measure of diffusion is the *diffusion flux* J. In a macroscopic sense J is the amout of mass diffusing through and perpendicular to a unit cross-sectional area of solid per unit of time. In microscopic sense J is the number of atoms diffusing. The units of J are $\frac{kg}{m^2 s}$ or $\frac{atoms}{m^2 s}$. In a simple case of the diffusion in one direction with constant rate, the J can be expressed by Fick's first law

$$J = -D\frac{dC}{dx} \tag{1}$$

where D is the diffusion coefficient and C is the concentration of diffused atoms. The diffusion coefficient D and the exact format of the equation for the diffusion flux J are dependent on the type of the diffusion process. Equation 1 applies to a process where gas is diffusing through a thin metal plate, so that the pressure of the gas is constant on both sides of the plate. (Callister & Rethwisch 2014).

The smaller the D coefficient is, the slower is the diffusion, i.e., the less mass/atoms are transporter per time unit. D is dependent on temperature according to equation (Callister & Rethwisch 2014)

$$D = D_0 e^{-\frac{E_A}{k_B T}} \tag{2}$$

where D_0 is a temperature-independent pre-exponential factor, E_A is the activation energy of the diffusion process, k_B is Boltzmann constant and T is the absolute temperature. The activation engery E_A can be thought as an energy required to produce the diffusive motion of one mole/kilogram/piece of atoms. A large activation energy results in a small diffusion coefficient. Pre-exponential factor D_0 can be thought as number/mole of atoms that are in a position to be able to do diffusive motion. The exponential term $e^{-\frac{E_A}{k_B T}}$ term gets value from 0 to 1 and it can be thought as a probability of atoms to do diffusive motion.

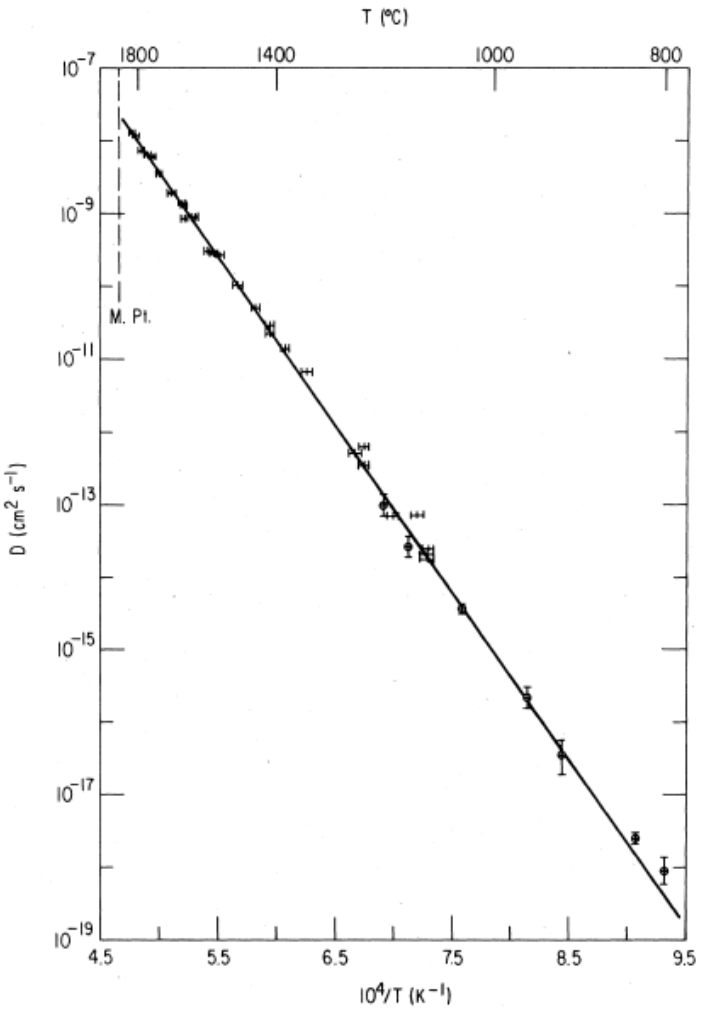

Figure 2: A plot for self-diffusion in chromium (Mundy 1981)

The way to explore diffusion coefficient D is to take a logarithm of equation 2.

$$ln\,D = ln\,D_0 - \frac{E_A}{k_B}\left(\frac{1}{T}\right) \quad (3)$$

This is the form of a straight line $y = b + mx$.

The figure 2 is an example of measurements following equation 3. The figure plots diffusion coefficient of self-diffusion in chromium in different temperatures. The curve in the figure is a straight line. The slope of the line is $-\frac{E_A}{k_B}$ providing a way to determine activation energy. The intercept of the line and y -axis tells logarithm of the pre-exponential factor $\log D_0$. This is the manner in which the values of E_A and D_0 are determined experimentally (Callister & Rethwisch 2014). When simulating diffusion coefficient, the results should form a similar straight line if the simulation works properly.

The type of diffusion studied in this thesis is surface diffusion. In surface diffusion there is an adatom on the top of the lattice surface changing place due to energy of thermal vibration. Like atoms in solids, also an adatom locates in coordinates defined by the lattice parameters. The adatom locates in one of the places where atoms of the next layer of lattice above the surface would locate. These lattice locations are energetically favored and there is the energy barrier between adjacent lattice locations. The adatom needs to have sufficient energy in order to overcome the energy barrier and move to the neighboring lattice location on the surface. Compared to the diffusion inside the solid, the surface diffusion is faster, for the surface diffusion there are always multiple, empty, adjacent neighboring sites for the adatom. With diffusion inside solid the diffusing atom needs to find space between other atoms or exchange place with other atoms.

Surface diffusion will follow the same temperature dependency as other types of diffusions. But because the surface diffusion is studied on the level of individual adatom it is more meaningful to write equation 2 based on reaction rate than based on diffusion coefficient, which is based on the amount of mass transported in the diffusion. Temperature dependency of reaction rate can be written as

$$\Gamma = \Gamma_0 e^{-\frac{E_A}{k_B T}} \quad (4)$$

This is the Arrhenius equation (Britannica Academic 2020). Here Γ is the reaction rate expressed in Hertz ($1/s$), Γ_0 is pre-exponential factor, E_A is activation energy, k_B is Boltzmann's constant and T is absolute temperature. The temperature dependency is exactly the same format than with diffusion coefficient. Also, in case of reaction rate, it is possible to draw a plot like the one showing dependency between temperature and diffusion coefficient (figure 2). With the reaction rate this plot is called Arrhenius plot. Arrhenius plot shows the logarithm of reaction rate against inverse of the temperature.

The surface diffusion can occur in several ways. For adatoms the main diffusion mechanism are hopping and exchange. In hopping mechanism an adatom jumps over the saddle point of the potential barrier between two adjacent sites and migrates to the next one. In exchange mechanism an adatom moves to the position of a surface atom which becomes adatom at the next-nearest binding site. G. A. Evangelakis and N. I. Papanicolaou have simulated the surface diffusion of Cu adatoms on Cu(001) surface in a temperature range from 700 K to 1 100 K. They found out that in addition to hopping and exchange processes there were multiple hopping events (an adatom moves directly a distance more than one binding site) and complicated multi-particle exchange processes. According to their simulations the duration of hopping event is about 0.5 ps and the duration of simple exchange process about 1.0 ps. (Evangelakis & Papanicolaou 1996).

Also this thesis focuses on surface diffusion of Cu adatoms on Cu(001) surface.

2.2 Simulating diffusion on metals with classical molecular dynamics (MD)

Classical molecular dynamics (MD) is a well-established simulation method used in many physical applications. A good definition of MD is given by Steve Plimpton: "Classical molecular dynamics (MD) is a commonly used computational tool for simulating the properties of liquids, solids and molecules. Each of the N atoms or molecules in the simulation is treated as a point mass and Newton's equations are integrated to compute their motion. From the motion of the ensemble of atoms a variety of useful microscopic and macroscopic information can be extracted such as transport coefficients, phase diagrams and structural or conformational properties. The physics of the model is contained in a potential energy functional for the system from which individual force equations for each atom are derived." (Plimpton 1995).

As shown by Arrhenius equation 4 the diffusion slows down rapidly when temperature decreases. The lower the temperature the smaller is the portion of velocity distribution that exceeds kinetic energy needed to overcome the potential barrier between adjacent lattice locations. In order to see diffusion events in MD simulation the simulation needs to have so many simulation steps that some of them are in the area of velocity distribution where the diffusion can happen. More timesteps are covered when the simulation runs longer, which means more computer time is needed. Same requirement for long computer runs, as for the diffusion, applies in MD simulation also to other infrequent processes where interesting events happen beyond the microsecond time scale.

When simulating infrequent events, like the diffusion in low temperatures, results could be gotten faster if the rate of interesting event could be accelerated. In this thesis one that kind of method, CVHD, is presented. Using acceleration, it is possible to get more interesting events with same amount of computer time. The acceleration used in CVHD modifies the potential energy surface derived from the potential energy functional used in the simulation by adding a boost potential in regions close to the local minima, such that all transition rates are increased while relative rates are preserved (Miron 2003).

A commonly used potential model for metals used in MD simulations is the embedded-atom model (EAM) potential. Also, in this thesis EAM is used when simulating diffusion on copper surface. Theoretical background for EAM potential energy model comes from quantum mechanical density functional theory. (LeSar 2013).

The generalized form of EAM potential energy function is

$$U = \sum_i F_i \left[\sum_{j \neq i} f_{ij}(r_{ij}) \right] + \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \Phi_{ij}(r_{ij}) \qquad (5)$$

here f is some function of interatomic distance representing an approximation of the electron density and Φ is a pair potential. Function F has its arguments as a sum over functions f_{ij} that depend on local positions of the atoms.

It is possible to write EAM potential energy function using local electron density $\bar{\rho}_i$, which is evaluated at the site of atom i and is approximated as a sum of contributions of the electron densities $\bar{\rho}_i(r_{ij})$ from atoms that neighbor atom i

$$\bar{\rho}_i = \sum_{j \neq i} \bar{\rho}_i(r_{ij}) \tag{6}$$

then functional F can be chosen to represent the energy to embed an atom i in a uniform electron gas of density $\bar{\rho}_i$. EAM potential energy function can then be written as

$$U = \sum_i F_i(\bar{\rho}_i) + \frac{1}{2}\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \Phi_{ij}(r_{ij}) \tag{7}$$

The task of fitting EAM parameters is then to find the forms of $F, \bar{\rho}_i$ and Φ that correspond to experimental results. This is done iteratively, first selecting some set of $F, \bar{\rho}_i$ and Φ, executing MD simulation, comparing results to experiments and adjusting parameters until calculated parameters fit the experimental values as closely as possible. In practice, it is not possible to present $F, \bar{\rho}_i$ and Φ as analytical functions but they are tabulated numerically as a parameter of r_{ij} until to some cut-off distance. Thus, the EAM potential energy function is in practice a table of values for $F, \bar{\rho}_i$ and Φ with different r_{ij}.

In MD there are two well-known method for calculating diffusion coefficient D. The first method is based on mean squared displacement and the second method is based on the velocity autocorrelation function. The relation between mean squared displacement and the diffusion coefficient was first derived by Einstein. It can be written as follows (Frenkel & Smit 2002).

$$\frac{\partial \langle r^2(t) \rangle}{\partial t} = 2dD \tag{8}$$

where d is the dimensionality of the system. $\langle r^2(t) \rangle$ is the mean squared distance that atoms have moved since time origin. If $\boldsymbol{r_i}(t)$ is a position of particle i at time t and $\boldsymbol{r_i}(0)$ is a position of the same particle at time $t = 0$ then mean square displacement for MD simulation of N particles can be written as

$$\langle r^2(t) \rangle = \langle |\boldsymbol{r}(t) - \boldsymbol{r}(0)|^2 \rangle = \frac{1}{N}\sum_{i=1}^{N} \left(\boldsymbol{r_i}(t) - \boldsymbol{r_i}(0)\right)^2 = \frac{1}{N}\sum_{i=1}^{N} \Delta r_i(t)^2 \tag{9}$$

The displacement of the particle i at time t, $\Delta r_i(t)$, can be written using time integral of the velocity of the particle

$$\Delta r_i(t) = \int_0^t dt'\, v_i(t') \tag{10}$$

In one dimension equation 8 can be written as

$$2D = \lim_{t\to\infty} \frac{\partial \langle x^2(t)\rangle}{\partial t} \tag{11}$$

where $x(t)$can be written as a time integral of the x component of the velocity v_x

$$\langle x^2(t)\rangle = \left\langle \left(\int_0^t dt'\, v_x(t') \right)^2 \right\rangle$$

$$= \int_0^t \int_0^t dt'\, dt''\, \langle v_x(t') v_x(t'')\rangle$$

$$= 2\int_0^t \int_0^{t'} dt'\, dt''\, \langle v_x(t') v_x(t'')\rangle \tag{12}$$

where $\langle v_x(t')v_x(t'')\rangle$ is the velocity autocorrelation function. The velocity autocorrelation function measures the correlation between the velocity of a particle at times t' and t''. In other words, it describes correlations between velocities at different times along an equilibrium trajectory. Therefore, the velocity autocorrelation function is an equilibrium property of the system. As an equilibrium property the velocity autocorrelation function is invariant under a change of the time origin and depends only on the difference of t' and t''. By defining $\tau \equiv t' - t''$ it can be written

$$\langle v_x(t')v_x(t'')\rangle = \langle v_x(t'-t'')v_x(0)\rangle = \langle v_x(\tau)v_x(0)\rangle \tag{13}$$

combining equations 11, 12 and 13 give

$$2D = \lim_{t\to\infty} 2\int_0^t dt''\langle v_x(t'-t'')v_x(0)\rangle$$

$$D = \int_0^\infty d\tau\, \langle v_x(\tau)v_x(0)\rangle \tag{14}$$

Equation 14 is the second well-known method in MD for calculating diffusion coefficient, this method is based on velocity autocorrelation function and is equivalent with the first, mean squared

displacement based method (Frenkel & Smit 2002). Most MD simulation software have built-in capabilities for calculating both mean squared displacement and velocity autocorrelation function.

The reaction rate can be obtained from MD simulation by analyzing trajectories of simulated particles. For the trajectory of the particles MD simulation software needs to write the location of a particle to a file (dump file) with suitable intervals. Using the dump file, it is possible to follow the trajectory of a single particle. For adatom diffusion, it is possible to follow the trajectory of the adatom and count how many times the adatom “hops” to another lattice site. If the adatom participates in an exchange event, then the trajectory analysis needs to be continued with the new adatom.

3 Theory of the CVHD methodology

Collective variable-driven hyperdynamics (CVHD) is a methodology that is aimed to accelerate MD simulation of slow processes, like diffusion. CVHD works by changing, boosting, potential energy function used in MD simulation so that events under the interest happen more frequently than with the original potential energy function. The boost is implemented by adding a suitable bias to the potential energy function, which fills potential energy minima and thus shortens waiting time between minima-to-minima transitions. The construction of an optimal biased potential energy function is not a trivial task. CVHD borrows concepts from hyperdynamics and metadynamics in order to construct well working biased potential. (Bal & Neyts 2015a).

When CVHD is used, the events under the interest happen with a smaller number of simulation timesteps than without CVHD. Because physical interpretation of reaction rates for accelerated and non-accelerated simulation should be the same, it can be thought, that in CVHD simulation times goes faster so that increased number of events divided by accelerated time gives correct reaction rate. The faster moving time means that CVHD covers longer timespan with same number of simulation steps than normal MD. In that sense CVHD extends timescale of the simulation. The idea is to model long-time evolution while still accounting for the underlying short-time processes as accurately as possible. Time scales between short-time processes and long-time evolution may be separated by several orders of magnitude, from fast atomic vibrations to slow structural changes (Miron 2003).

Hyperdynamics in MD is a methodology that is based on the idea of adding a bias, ΔV, to the potential energy function when the system is far from the transition state and ensuring that ΔV goes to zero in the transition state region. The condition $\Delta V = 0$ in the transition state region preserves the correct relative dynamics. In the context of this thesis, the transition state is the state of the system where an atom diffuses from one location to another. Hyperdynamics adds potential energy when the atom to be diffused is near the bottom of the potential energy well and thus increase the probability that the atom moves up in the well. When the atom is near the edge of the potential energy well and is about to move to another location then original potential energy function is used which ensures the correct relative dynamics. (Voter 1997).

Metadynamics is a method that was designed to explore reaction pathways and calculate free energy surfaces. Metadynamics uses history-dependent bias for the potential energy. A key concept in metadynamics is a collective variable (CV) which is formed from the quantities describing the properties of the system. Number of CVs in a system should be small and it should be possible to distinguish between all the relevant states of the system using CV(s). Number of CVs is very much less than number of atomic positions in the simulated system and thus few CVs can be sampled extensively to calculate statistical quantities accurately. Also, low number of CVs might ease comparison with experiments or construction of empirical models. (Fiorin & Klein & Hénin 2013).

In metadynamics the bias to be added to the potential energy function is constructed as a function of CVs. This keeps calculation of the bias simple. The bias additions should be sufficiently slow, so that those degrees of freedom, that are not included in CV, would have enough time to equilibrate and this need to be taken into account in the form of bias construction function. Unlike in hyperdynamics, in metadynamics, the bias is applied also in the transition state region. CVs introduced in metadynamics are used in CVHD but following hyperdynamics, CVHD doesn't apply the bias near the transition state.

When metadynamics is used to calculate free energy surfaces then metadynamics potential V_{meta} is constructed as a sum of repulsive Gaussian hills. The dimensionality of the hills corresponds to number of CVs. The hills are centered around values of CVs in each timestep of the simulation. This way the metadynamics potential is history-dependent, at time t the metadynamics potential V_{meta} is a sum of previous values of V_{meta}. (Laio & Parrinello 2002).

In metadynamics simulation the effective potential V_{eff} is the sum of the real underlying potential V, which is derived from potential energy functional used in MD simulation, and V_{meta}. These potentials can be written as a function of collective variable η.

$$V_{eff}(\eta) = V(\eta) + V_{meta}(\eta) \tag{15}$$

During the simulation, the system evolves towards the nearest minimum of $V_{eff}(\eta)$ and at same time new Gaussian hills are added centered at value of CV, η, corresponding to that minimum. When the simulation proceeds, potential wells are filled with Gaussian hills. After sufficient time each minimum is cancelled out and at that stage V_{eff} is constant. The free energy surface can then be obtained from equation 15.

$$V(\eta) \cong -V_{meta}(\eta) + K$$

where K is an additive constant and V_{meta} consist of Gaussian hills calculated during the simulation.

CVHD has a modular structure. The CVs, on which bias is based, can be selected independently from the biasing algorithm itself. Thus, one biasing algorithm can be used with several different CVs and vice versa. This makes it possible to select the most suitable CVs and biasing algorithm for each system. This modularity makes the CVHD very versatile. Based on the type of biasing algorithm, the CVHD methodology can be divided to statically biased CVHD (sCVHD) and to dynamically biased CVHD (dCVHD). In sCVHD the biasing algorithm is a static, analytical, function of CVs and the function doesn't change during the simulation run. In dCVHD a suitable history-dependent bias is calculated on-the-fly during the simulation run. The history-dependent bias is borrowed from metadynamics into dCHVD. Because the bias evolves in dCVHD it can be thought that dCVHD is one type of self-learning method. (Bal & Neyts 2015a).

In hyperdynamics, and thus in CVHD, the simulations are performed on the modified potential energy surface, $V^*(\boldsymbol{R})$, not on the true potential energy surface $V(\boldsymbol{R})$. $\boldsymbol{R}$ represents here the coordinates of all particles in the simulated system. (Bal & Neyts 2015a).

$$V^*(\boldsymbol{R}) = V(\boldsymbol{R}) + \Delta V(\boldsymbol{R}) \tag{16}$$

where $\Delta V(\boldsymbol{R})$ is the bias potential energy function. In CVHD the bias potential $\Delta V(\boldsymbol{R})$ is reduced to a function of only one parameter, collective variable η. It can be thought that η is a global reaction coordinate. In CVHD η can have continuous values between 0 and 1. The fact that $\Delta V(\boldsymbol{R})$ can be written as $\Delta V(\eta)$ is the key simplification of CVHD (Bal & Neyts 2015a). The equation 16 can then be written as

$$V^*(\boldsymbol{R}) = V(\boldsymbol{R}) + \Delta V(\eta) \tag{17}$$

3.1 BondBreak - Collective variable for CVHD in simulation of the diffusion

For CVHD a collective variable, CV, based on local system properties, is developed so that the CV is dependent on how much each property contributes on the actual transition under the interest. The idea is not to treat all properties equally, but to strongly emphasize those properties that are involved in the actual transaction. The bias can then be adjusted based on how close to the transition state the CV is. The closer to transition value the CV is, the stronger bias is driven to zero.

In the case of the diffusion, as in this thesis, the relevant local system property is the bond length, the distance between atoms in pair i, r_i. For an atom to diffuse, it must first break the bonds with neighboring atoms, then move to a new position and create new bonds with new neighboring atoms. The bond is broken between two atoms when the distance between them is large enough and similarly a new bond is created when atoms are close enough to each other. Thus, bond length can be used as a measure, if there exist a bond between atoms in atom pair i. Using bond length, i.e. r_i, as a collective variable the bond is broken when r_i exceeds the distance after which atoms in pair i has no effect on each other. In case of the simulation in solids, the suitable fraction of lattice constant can be used as a maximum distance for the bond. In the implementation of CVHD used in this thesis, the CV based on interatomic distances and bond breaks is called *BondBreak* CV.

In order to be physically meaningful, the CV should be invariant under global translations and rotations (Fiorin & Klein & Hénin 2013), this is true for *BondBreak* CV. Interatomic distances are independent of translations and rotations of coordinate system.

With other processes, the relevant system property to be used as a CV, can be for example dihedral angle between bonds, coordination number CN, root mean square displacement from reference positions, angle of rotation around a given axis or change in the internal structure of a group of atoms. (Fiorin & Klein & Hénin 2013).

With the bond length as a CV, the length can be divided into 4 different regions using simulation/system specific limits r_i^{min}, r_i^{max} and r_i^{cut} (Bal & Neyts 2015a). $r_i^{min}$and r_i^{max} mark the shortest and the longest distance on which the bond is breaking. At the beginning of the simulation and after a bond break happens, r_i^{cut} is used to select atom pairs having so short mutual distance that there exists a bond, that could break, between the atoms. r_i^{min}, r_i^{max} and r_i^{cut} are given as input parameters of *BondBreak* CV and the logical relation between them is $r_i^{min} \leq r_i^{cut} \leq r_i^{max}$.

1. If the length r_i is greater than a cutoff r_i^{cut} then a stable bond between atoms does not exist and this atom pair is not followed. This criterion is applied only on relaxation periods, when the simulation starts and a bond break has happened. Based on this criterion, a list of bonds is created from all atomic pairs that are in a shorter distance from each other than r_i^{cut}. Later in the simulation following 3 criteria are applied only to that list of bonds.
2. If $r_i \geq r_i^{max}$ then the bond is dissociated or just about to dissociate. The diffusing atom is then very likely moving to a new position and thus the system is in a transition state region. According to principles of hyperdynamics the bias potential energy, ΔV, should then be zero.
3. If $r_i^{max} > r_i \geq r_i^{min}$ then the system is between the stable region, where the bond is not likely to dissociate soon, and transition state region. In this region ΔV is to be decreased the closer r_i is to transition region, r_i^{max}.
4. If $r_i^{min} > r_i$ then atoms are so close to each other that the bond between them is very unlikely to dissociate soon.

Based on the classification above it is possible to define a local distortion χ_i as follows for all atomic pairs i where atoms are a shorter distance from each other than r_i^{cut} (Bal & Neyts 2015a)

$$\chi_i = \begin{cases} 0, & r_i \leq r_i^{min} \\ \dfrac{r_i - r_i^{min}}{r_i^{max} - r_i^{min}}, & r_i^{min} < r_i < r_i^{max} \\ 1, & r_i \geq r_i^{max} \end{cases} \tag{18}$$

This local distortion, χ_i, gets a value of 0 when the bond in atom pair i is not to dissociate, value of 1 when the likelihood for the dissociation is large and a linear interpolation from 0 to 1 when the distance between atoms is between r_i^{min} and r_i^{max}.

Local distortion is calculated only for atom pairs that are in the list of bonds, this list is created at the beginning of the simulation and recreated each time when a bond break happens. The list of bonds saves computation time when it is not necessary to calculate the local distortion for all atom pairs. Initially the list of bonds contains atom pairs where atoms are a shorter distance from each other than r_i^{cut} at the beginning of the simulation. When a bond break happens all atom pairs are checked in this new state in order to find atom pairs where atoms are in a shorter distance from each other than r_i^{cut} and these atom pairs are placed in the recreated list of bonds. The calculation of the local distortion continues then again only for atom pairs in the newly created list of bonds, until a new bond break happens, and the list of bonds is recreated again.

The selection of r_i^{min} and r_i^{max} affects speed and the quality of CVHD simulation. The larger r_i^{min} and r_i^{max} are, the more bias, ΔV, is introduced on the system and the faster simulation proceeds. But on the other hand, if r_i^{max} is too large, then ΔV might be applied in the transition state region violating the requirement of hyperdynamics. And similarly, if r_i^{min} is too large, it might be that ΔV is not decreased when it should be. So, although large values of r_i^{min} and r_i^{max} speed up the simulation, too large values lead to incorrect dynamics, and thus some safety margin need to be applied when selecting r_i^{min} and r_i^{max}.

Using local distortion, χ_i, it is possible to define global distortion, χ_T, using the formula (Bal & Neyts 2015a)

$$\chi_T = \left(\sum_i^N \chi_i^p \right)^{1/p} \tag{19}$$

Global distortion describes how close to the transition state region the system is. The greater the value of χ_T is the closer to the transition state region the system is. Because local distortions have values between 0 and 1 (including), it means that global distortion is a positive real number. N is here number of atom pairs having a mutual distance shorter than r_i^{cut}, not the total number of atoms in the simulation.

The parameter p in equation 19 ensures that large distortions make a larger contribution to χ_T than small ones. The parameter has a restriction $p > 1$. The greater p is, the smaller the contribution of small local distortions to χ_T are. Because χ_T is used to determine bias of the potential energy, the large value of p then means that bias is controlled by large local distortions leading to more localized bias than a small value of p. In other words, p controls how bias energy is distributed across the system. According to (Bal & Neyts 2015a) p is not a critical parameter for the CVHD method and its effect was found to be rather small for values between 4 and 12.

In principle equation 19 approaches the max function when $p \to \infty$. This would mean that when CV is the distance between atom pairs, then with max function, only the longest distance between atoms among all atom pairs would have a contribution to CV. So, with very large value of p, the longest inter-atomic distance would dominate the CV. However also in the case of large p, it is necessary to count CV using all atom pairs, because it cannot be said beforehand which atom pair has the longest distance and also the pair having the longest interatomic distance might change during the simulation. Using $\max(r_i)$ as CV wouldn't save computing time by decreasing the number of atom pairs that need to be checked but would save some computing time because calculating $\max(r_i)$ is computationally faster than calculating equation 19. The derivatives of collective variables calculated in CVHD should be continuous and this is not necessary the case for $\max(r_i)$ and therefore more elaborate forms should be used for *BondBreak* CV despite of appealing simplicity of $\max(r_i)$ (Bal 2018).

As a side note, the equation 19 can be generalized to a case where one wants to follow n bond to break instead of 1. This can be done by dividing χ_i by number of bondbreaks, n, we are interested in. Equation 19 has then form

$$\chi_T = \left(\sum_i^N \frac{\chi_i^p}{n} \right)^{1/p} \tag{20}$$

and $\chi_T$will become to 1 if n of local distortions, χ_i, are 1 (which is the case we are interested in) or if plenty enough of local distortions differ from 0.

In CVHD the CV is required to have values in a range $[0,1]$. Hyperdynamics requires CV to have continuous and vanishing derivatives when CV goes to 0 or 1, because CV is used to calculate bias

potential $\Delta V(\boldsymbol{R})$, and the negative gradient of $\Delta V(\boldsymbol{R})$ is the biasing force applied to the atoms in simulations and in hyperdynamics the biasing force should vanish in transition region (Miron 2003). The global distortion χ_T does not fulfill these requirements and therefore the actual CV, η, is calculated from χ_T according to (Bal & Neyts 2015a)

$$\eta = \begin{cases} \frac{1}{2}(1 - \cos(\pi\chi_T^2)), & 0 \leq \chi_t \leq 1 \\ 1, & \chi_T > 1 \end{cases} \tag{21}$$

η is limited in the range [0,1]. Value $\eta = 1$ (and $\chi_T \geq 1$) means that the system is in a transition state range and biasing force is not applied to the system. From figure 3 it can be seen that η smoothens both near 0 and near 1. This means that biasing force deduced from the derivative of η goes to zero near 0 and 1 as it should do. (Bal & Neyts 2015a).

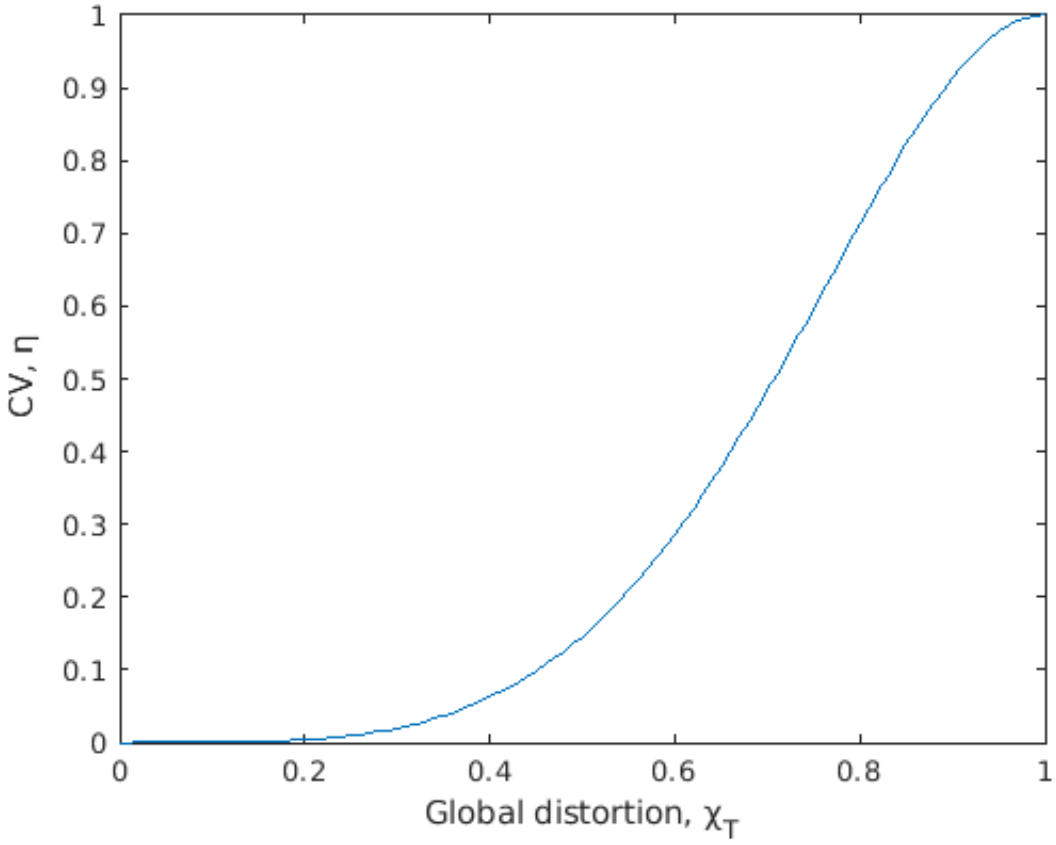

Figure 3: A plot of η

This format of η allows to describe transitions involving multiple local distortions in addition to describing a transition involving a single distortion - $\chi_T \geq 1$ if one $\chi_i = 1$ or if all N χ_is are at least $(1/N)^{1/p}$. This also means that a single value of χ_T (and thus η) correspond to more than one state of the system. Therefore, in CVHD it is necessary to drop the requirement of metadynamics to distinguish the different states of the system by the value of CV. CVHD, however, works without fulfilling this requirement. (Bal & Neyts 2015a).

For other processes than diffusion it is quite easy to modify equation 18 to use properties specific for the process in interest, for example dihedral angles instead of bond length. The formulation of η used in CVHD is thus rather generic.

3.2 Biasing algorithm

In statically biased CVHD, sCVHD, the biasing algorithm is a static, analytical, function of η. The simplest way to build bias potential in sCVHD is to use a linear function of η

$$\Delta V(\eta) = \Delta V^{max}(1 - \eta), \tag{22}$$

where ΔV^{max} is the maximal bias strength. When inserted in equation 16 this leads to

$$\mathrm{V}^*(\mathbf{R}) = \mathrm{V}(\mathbf{R}) + \Delta V^{max}(1 - \eta) \tag{23}$$

ΔV^{max} must be chosen appropriately. It needs to be large enough to make a substantial change in the potential energy but not larger than the barrier of interest. Also r_i^{max} in equation 18 needs to be selected carefully so that η goes to 1 and ΔV^{max} to zero in the transition state range. This is also a main limitation of sCVHD, some a priori knowledge of the possible events in the system is needed, in order to define suitable ΔV^{max}.

In dynamically biased CVHD, dCVHD, the biasing algorithm used is borrowed from metadynamics. The algorithm will grow $\Delta V(\eta)$ at intervals τ_G in the form of Gaussian functions with width σ and height w (Bal & Neyts 2015a). The computational solution is to store η, σ and w in memory at intervals τ_G. $\Delta V(\eta)$ is then computed by adding together value of each previously stored Gaussian function at point η.

$$\Delta V(\eta) = \sum_{k<n_g} w_k \exp\left[-\frac{\left(\eta - \eta(k\tau_g)\right)^2}{2\sigma^2}\right] \quad (24)$$

where n_g is a number of times the growing step has been applied. The resulting biasing potential is thus a sum of repulsive Gaussian functions (Fiorin & Klein & Hénin 2013). Because $\Delta V(\eta)$ is constructed by summing together all previous additions to $\Delta V(\eta)$ the bias potential is history dependent. Because the bias is repulsive, it will drive particles farther from each other, and thus in the case of diffusion the diffusion phenomenon is accelerated. It is possible, that during the simulation, the system has same value for η at two different point of time. Due to history-dependency $\Delta V(\eta)$ might however be different for the same value of η at different times, so to be exact ΔV should be written as a function of both η and n_g, $\Delta V(\eta, n_g)$.

This algorithm will, by default, keep adding the bias at any time therefore it might be necessary to restrict the algorithm from adding the bias at large η values, for example at $\eta > 0.9$.

Essentially, both in sCVHD and dCVHD, the binding energy of the system at equilibrium is lowered by adding a small repulsive potential to bonds like in bond-boost method (Miron 2003).

In order to be able to describe multiple consecutive events the biasing algorithm needs to reset ΔV after each transition. The way CVHD recognizes a transition is to follow during a waiting time t_w that η remains equal to 1. This ensures that the transition is stable, situations where the system moves near to or little over a saddle point and returns to earlier state are not recorded as transitions. After the transition system will be thermalized in its new state, a new η is calculated and CVHD procedure is resumed. With dCVHD this means that all accumulated Gaussian hills are deleted. (Bal & Neyts 2015a). CVHD lets the system to stabilize time t_w after the transition. This means that after transition there is a period of $2t_w$ when bias is not applied. The waiting time t_w is a parameter for CVHD. The suitable value of t_w relates to the duration of the simulated event and sufficient safety margin for not counting back and forth movements as transitions.

This “reset” functionality makes a difference between dCVHD and metadynamics. In dCVHD the accumulated bias potential is not stored from one state to another like in metadynamics. In dCVHD when the system moves to a new state all accumulated bias potential is deleted and the bias

calculation is restarted from zero. In metadynamics the bias in a new state is accumulated on the top of the bias accumulated in the earlier states so that bias eventually fills all local minima.

sCVHD requires an analytical function for calculation of the bias potential and thus a priori knowledge of the processes in the system is needed. dCVHD builds the bias on-the-fly and thus dCVHD can adapt to processes with unknown activation barrier as long as system dynamics can be presented by η. sCVHD will build up the bias faster in the beginning than dCVHD that first needs to learn how to build the bias. sCVHD is also easier to parametrize because it doesn't require Gaussian parameters like dCVHD. The Gaussian algorithm of dCVHD will use dynamically larger bias if the waiting time between successive events is long. For these reasons sCVHD could be more suitable if the studied system is well-characterized or in a case of relatively fast successive events. On the contrary dCVHD, could be more suitable if the studied system is not well known or in a case of long waiting time between successive events.

In MD simulation the force is derived as a gradient of the potential energy. CVHD uses a biased potential energy $\mathrm{V}^*(\mathbf{R})$ from equation 17 and by differentiating it the force is

$$F(\boldsymbol{R}) = -\nabla V^*(\boldsymbol{R}) = -\nabla[\mathrm{V}(\mathbf{R}) + \Delta\mathrm{V}(\eta)] = -\nabla\mathrm{V}(\mathbf{R}) - \frac{\partial\Delta\mathrm{V}(\eta)}{\partial\eta}\nabla\eta \tag{25}$$

For the sCVHD the gradient of $\Delta\mathrm{V}(\eta)$ is simply ΔV^{max} as can be seen from equation 22. For dCVHD the gradient of $\Delta\mathrm{V}(\eta)$ can be calculated as sum of gradients of individual Gaussian functions in equation 24. The gradient of CV, $\nabla\eta$, can be calculated for each atom using derivatives of functions 18, 19 and 21 for local distortion, χ_i, in range [0,1]. Outside of range [0,1] χ_i is constant, its gradient is 0 and there is no biasing force. The gradient per atom pair i, $\nabla\eta_i$, in range [0,1] is

$$\nabla\eta_i = \frac{\pi\chi_T \sin(\pi\chi_T^2)\left(\sum_i^N \chi_i^p\right)^{1/p-1}\chi_i^{p-1}}{r_i^{max} - r_i^{min}}\hat{\boldsymbol{r}}_{\boldsymbol{i}} \tag{26}$$

where $\hat{\boldsymbol{r}}_{\boldsymbol{i}}$ is a unit vector along the distance between atoms in atom pair i. The total gradient for a single atom is sum of gradients of all atom pairs where the atom belongs to.

When parametrizing the bias potential (ΔV^{max}, σ, w, τ_G and t_w), the same restriction and safety measures should be considered as with the bond-boost method. For preserving the accuracy of the

numerical integration, the biasing forces (gradients of the bias potential) should be on the same scale as the natural forces in the system. This implies that the magnitude of the bias potential should be comparable to the natural energy barriers in the system. Numerical integration inaccuracies may appear if the boost potential induces too high derivatives because of including too steep curves. Too large bias potential might induce local peaks within the local potential basin. These peaks could create a repulsive region close to the original local minimum, and the system only infrequently penetrates the high-boost region, which decreases the efficiency of boosting. (Miron 2003).

3.3 Boost achieved with CVHD

Using transition-state theory, TST, formalism, the TST rate $k_{i\to j}^{TST}$ of a particular transition (an adatom diffusion event in the context of this thesis) is given by the flux through the dividing hypersurface that separates states i and j (Miron 2003), (Voter 1997)

$$k_{i\to j}^{TST} = \frac{1}{2}\frac{\iint d\boldsymbol{r}\, d\boldsymbol{p}\, \delta_{ij}^{\perp}\boldsymbol{\Theta}_i|v_{\perp}| \exp\left(-\frac{K+V}{k_B T}\right)}{\iint d\boldsymbol{r}\, d\boldsymbol{p}\, \boldsymbol{\Theta}_i \exp\left(-\frac{K+V}{k_B T}\right)} \tag{27}$$

where k_B, T, K and V are respectively Boltzmann constant, temperature, kinetic and potential energies. $\boldsymbol{\Theta}_i$ is the occupation function which is 1 if the system is in state i and zero otherwise. $\delta_{ij}^{\perp}$ is the delta function defining the location of the dividing hypersurface, and $|v_{\perp}|$ is the velocity component orthogonal to the hypersurface.

TST assumes that the system equilibrates in each local minimum before jumping away. The transition rates out of each minimum are then equilibrium quantities. MD follows the short-time-scale Newtonian dynamics of the system, thus the TST rates arise naturally, albeit they may be too slow for any significant process to happen over the time probed in a typical simulation without acceleration like CVHD (Miron 2003). When a surface diffusion event (adatom hop or exchange) happens, the adatom is equilibrated in a new location in the lattice surface before the next diffusion event occurs. The TST formalism is thus suitable for describing reaction rates of surface diffusion events covered in this thesis.

CVHD uses the biased potential energy $\mathrm{V}^*(\mathbf{R})$ from equation 17. In this biased potential the bias potential $\Delta\mathrm{V}(\eta)$ goes to zero on the dividing hypersurface states i and j. By substituting equation 17

in equation 27 and taking in to account that $\Delta V(\eta)$ goes to zero on the dividing hypersurface only the denominator changes. The boosted transition rate $k_{i\to j}^{b}$ is thus (Miron 2003)

$$k_{i\to j}^{b} = \frac{1}{2}\frac{\iint d\boldsymbol{r}\, d\boldsymbol{p}\, \delta_{ij}^{\perp}\boldsymbol{\Theta}_i |v_\perp| \exp\left(-\frac{K+V}{k_B T}\right)}{\iint d\boldsymbol{r}\, d\boldsymbol{p}\, \boldsymbol{\Theta}_i \exp\left(-\frac{K+V+\Delta V(\eta)}{k_B T}\right)} \tag{28}$$

The denominator is smaller and $k_{i\to j}^{b}$ is greater than $k_{i\to j}^{TST}$ if $\Delta V(\eta)$ is positive, which is guaranteed by the construction of $\Delta V(\eta)$ both in sCVHD and dCVHD.

$$\frac{k_{i\to j}^{b}}{k_{i\to j}^{TST}} = \left\langle \exp\left(\frac{\Delta V(\eta)}{k_B T}\right)\right\rangle_b \tag{29}$$

where $\langle \ldots \rangle_b$ represents a canonical ensemble average done on the boosted potential surface (Miron 2003). The average time between events leading from state i to state j, $\tau_{i\to j}$, is the inverse of the corresponding reaction rate

$$\tau_{i\to j}^{TST} = \tau_{i\to j}^{b}\left\langle \exp\left(\frac{\Delta V(\eta)}{k_B T}\right)\right\rangle_b \tag{30}$$

So, the TST based physical time-span $\tau_{i\to j}^{TST}$ covered by boosted simulation time $\tau_{i\to j}^{b}$ becomes a statistical average. The ratio between $\tau_{i\to j}^{TST}$ and $\tau_{i\to j}^{b}$ is the boost achieved by CVHD. Thus the acceleration achieved with CVHD is an ensemble average of the biased potential energy surface (Bal & Neyts 2015a)

$$Boost = \left\langle e^{\frac{\Delta V(\eta)}{k_B T}}\right\rangle \tag{31}$$

$Boost$ is the acceleration factor relative to standard MD. The effective simulated time, *hypertime*, is calculated by multiplying the MD time with $Boost$ (Voter 1997). For example, if $Boost$ is 10^6 then MD simulation of 1 ns will correspond to a hypertime of 1 ms. Because ΔV is zeroed in CVHD when the system moves from one state to another and ΔV is built independently in each state, the $Boost$ can be different in the different states of the simulation. With CVHD constant $Boost$ cannot be ensured in a long simulation which involves many state transitions. The total *Boost* in CVHD

simulation is obtained by calculating *Boost* separately for each timestep and then adding *Boost* of each timestep together.

As can be seen from equation 31, the $Boost$ will decrease when temperature increases. This sets the temperature limit for the efficiency of CVHD. At a high enough temperature *Boost* becomes negligible.

There is a difference between sCVHD and dCVHD regarding the accuracy of the calculated hypertime. Accurate hypertime calculation requires good sampling of the regions with large bias potential energy. In the case of dCVHD, the large amount of the bias is accumulated with values of η that occur frequently during the simulation. The more often the simulated system visits certain range of η the more ΔV is accumulated on that region. So, with dCVHD good sampling of η space corresponds to large amount of the bias accumulated on that region of η leading to accurate hypertime calculation. With sCVHD there is no correlation of amount of the bias and the amount of time the simulated system spends on the different parts of η space. In sCVHD, ΔV is just a function of η, not the function of how often different values of η appears during the simulation. This limits some of the benefits of the sCVHD, because it does not produce as accurate hypertime calculation as dCVHD does. (Bal & Neyts 2015a).

3.4 Applicability and efficiency of CVHD

The CVHD algorithm will be efficient as long as slow events lead to a significant distortion of a small subset in a large collection of local parameters. On the contrary, if a process of interest only makes a small contribution to the CV, then CVHD would not be accurate and efficient. (Bal & Neyts 2015a). In the scope of this thesis, a diffusion event will lead to significant change of bond lengths associated with the diffusing atom and the changes in the bond lengths will then change collective variable η so that it reaches the value one. η based on bond length should therefore work well when using CVHD in the simulation of the diffusion.

CVHD utilizes concepts of metadynamics but there are differences between CVHD and metadynamics. Metadynamics is used to define dynamical pathways and to calculate free energy profiles by connecting a limited number of pre-known states when CVHD (and other accelerated MD methods) aims to find natural, unconstrained state-to-state dynamics over long time scales. Metadynamics is thus limited on a smaller part of phase space, which is sampled extensively, when

CVHD examines in the principle the full phase space. Technically this means that in CVHD it is not required that the different states of the system would be fully distinguished from each other using CV. (Bal & Neyts 2015a).

Because CVHD utilizes the hyperdynamics paradigm to set the bias potential energy ΔV to zero near the transition state region, there is a fundamental difference how the bias potential energy is accumulated in metadynamics and in CVHD. In metadynamics, the bias is accumulated during the whole simulation run and also in the transition state region. On the contrary in CVHD, the bias is set to zero in the transition state region and reset after the completed transition. In other words, CVHD accumulates bias independently for each period between successive transitions. The requirement to set ΔV to zero near the transition state region sets also a quality requirement for the CV and for the underlying local properties and distortions. The local distortions, χ_i, need to be defined so that they certainly go to 1 near the transition state. In the scope of this thesis, this means that r_i^{max} in equation 18 needs to be selected with a reasonable safety margin. This requires some prior knowledge of transitions, obtained for example, from earlier simulations.

The strength and the limitation of the CVHD method is that all relevant local system properties must be included in CV. When it is possible to describe the full dynamics by a simple CV, like chemical reactions including bond breaking and described by a CV based on the local distortion of the bond length, the CVHD methodology is suitable. On the other hand, if it is not possible to define a CV based on simple local system properties, like in a case of complex biological processes including various types of nonbonded interactions, then CVHD is impractical. The ability to build CV from local system properties and their distortion is thus a key to a successful use of CVHD.

An important disadvantage of CVHD is poor scaling with the system size. In the large system, events will occur more frequently, leading to additional overhead in resetting the bias and thermalizing the system. In large system, it is possible that η is close to 1 not because transition is about to happen in one part of the system, but because for many local distortions $\chi_i > (1/N)^{1/p}$. This leads η to go to 1 and the bias to disappear too far from the transition state leading to inefficient acceleration. In large enough system also, parallel events are possible and CVHD, that represents the dynamics by a single CV, does not allow parallel treatment of events. (Bal & Neyts 2015a)

Because CVHD adds potential energy to the system, it might introduce some unphysical heating when the added potential energy transforms to kinetic energy. Therefore, it is necessary to thermostat the simulated system. (Bal & Neyts 2015a).

As explained in chapter 3.3 the *Boost* achieved by CVHD is temperature dependent. The higher the temperature the smaller is *Boost*. This sets an upper limit for temperature, above which CVHD is not useful.

4 Implementation of CVHD in LAMMPS

4.1 LAMMPS software

The software framework selected for this thesis to run diffusion simulation is LAMMPS. LAMMPS is popular and well-known particle simulation code, developed and maintained at Sandia National Laboratories, USA. LAMMPS is based on the work of Plimpton (Plimpton 1995). LAMMPS stand for Large-scale Atomic/Molecular Massively Parallel Simulator. While LAMMPS is primarily aimed at Molecular Dynamics simulations of atomistic systems, it also provides a general, fully parallelized, framework for particle simulations governed by Newton's equations of motion. LAMMPS is customizable and there exist lot of add-on modules for LAMMPS. With suitable customization, it is possible to run CVHD simulations with LAMMPS.

LAMMPS works by reading in an input script, building the simulation model according to the input script and then running the simulation. LAMMPS input script has its own command language with variables and control structures (if, loop) which enables quite complex simulation scenarios. The LAMMPS command language is documented in LAMMPS manual. The structure of typical LAMMPS input script is as follows (Plimpton 2014)

- Definition of units used in input script
- Definition of simulation domain. LAMMPS has commands for creating atoms/particles, reading data from a file and creating simulation box with different boundary conditions. The particles can be joined in groups to form molecules or other structures
- Definition of properties of atoms; mass, velocity, charge and so on
- Definition of the potential energy functional used to simulate interactions between particles

- Applying constraints (most typically temperature and pressure but also more complex constraints are possible) and advancing to next time step (time integration). Constraints and time integration are defined using command `fix`. Therefore, applying constraints and doing time integration is called *fixing* in LAMMPS terminology.
- Definition of diagnostic calculations
- Definition of output
- Definition of the actual simulation run. Previous steps define the simulation model and only `run` command starts the actual simulation. For example, time integration, defined above, is executed only after the `run` command has introduced. While executing, the simulation run can be modified using LAMMPS control structures

The similar structure of the input script is also used when running diffusion simulations related to this thesis.

One key term in LAMMPS is *style*. *Style* defines the way a LAMMPS command works and what additional parameters are allowed. Styles are different for the different commands. For example, `atom_style` command defines what attributes atoms have. There are about 20 different styles for atoms defining whether charge, dipole moment, angles, density and other physical attributes are stored for atoms during the simulation. Similarly, there exist over 100 styles for different potential energy functionals to describe the different interactions between particles. These styles and the potential energy functional to be used are defined by command `pair_style`. Good examples of pair style are `lj/cut`, which is the basic Lennard-Jones potential with cutoff used in all introductory texts for MD, and `eam`, which is the potential energy functional used in this thesis to model interaction between copper atoms.

4.2 COLVARS module

The LAMMPS' distribution includes an optional module, named COLVARS, for using collective variables. A detailed explanation of its design is provided by Fiorin, Klein and Hénin (Fiorin & Klein & Hénin 2013). When building the LAMMPS executable, the COLVARS module must be compiled separately as a library and linked to the LAMMPS executable. The module includes more than 25 different types of CVs. In typical simulations the computational overhead of COLVARS module is negligible (Fiorin & Klein & Hénin 2013).

The processing of collective variables is based on a separate configuration file which is read by COLVARS module using the following line in LAMMPS input script

```
fix ID all colvars configfile keyword value pairs …,
```

where ID is a string that uniquely identifies this fix from others. Only requirement for ID is that it must be different than `ID` of any other fix command but of course it helps to use some meaningful `ID` like `CVfix1`. According to the general LAMMPS formalism, the second parameter for the `fix` command is name of group of atoms for which the fix is applied. The COLVARS module applies all its operations to the entire system and therefore the COLVARS module ignores the second parameter of `fix` command. For the sake of clarity, it is good to use built-in group `all` as a second parameter for `fix` command when using the COLVARS module. The third parameter in the fix command above is the *style* of the fix. In order to use the COLVARS module the style must be `colvars`. If the COLVARS module is not linked in the LAMMPS executable, then `colvars` is not recognized as an applicable fix style. The fourth parameter, `configfile`, is the name of the COLVARS configuration file. The rest of parameters for this `fix` command are optional keyword value pairs, that define some general properties of the COLVARS module: input and output files or their prefixes, thermostating method used, seed for random number generator and information how to take into account periodic boundaries in atom positions when calculating collective variables and the resulting forces. The `colvars` fix computes a global scalar which can be accessed by various output commands. The scalar is the cumulative energy change due to this fix.

The configuration file for the COLVARS module has three parts. The first part defines some general parameters that are used with all collective variables. These parameters define for example how often CV information is output and if symmetric multiprocessing (SMP) is used or not. The second part of the configuration file defines collective variables. One configuration file can contain definitions for the multiple collective variables. Each definition starts with keyword `colvar` and is enclosed in curly braces {}. The third and last part of the configuration file defines biasing and analysis methods that utilizes collective variables defined.

The biasing methods modify the simulated potential energy functional in order to make the simulation more effective. The biasing and analysis methods recognized by the COLVARS module included in LAMMPS distribution “LAMMPS 64-bit 16Mar2018” are adaptive biasing force (abf),

harmonic, histogram and metadynamics. In the configuration file, the definition of each of these methods is started with the name of method and definitions are enclosed in curly brackets {}. Definitions for collective variables and biasing and analyzing methods are multiline. The definitions in these three parts of the configuration file consist of keyword value pairs and might contain subdefinitions in the blocks enclosed in curly brackets {}.

In keyword value pairs, the keyword and its value are separated by any white space. Keywords are case-insensitive but string-based values are not. The COLVARS configuration file has same format even if the COLVARS module is linked to some other MD software than LAMMPS.

The basic output of COLVARS module is a "trajectory" of collective variable which lists values of collective variable at different steps of simulation. Additional output, for example for velocity and force, can be defined in the definitions of collective variables. Also biasing and analyzing methods can be configured to provide their own, method-specific, output files.

4.3 CVHD implementation

CVHD is built as an extension to the COLVARS module that itself is an optional addon module to LAMMPS. CVHD extension is written by Kristof M. Bal and Erik C. Neyts. The source code for CVHD extension is available free of charge (Bal & Neyts 2015b). The extension consists of C++ source and header files and instructions how to include them in build process of LAMMPS executable. The extension brings two new CV types in LAMMPS, `bondbreak` and `angleswitch`, one new biasing method `cvhd` and one new diagnostic calculation `timeboost`.

Both `bondbreak` and `angleswitch` implement equation 21. The CV of type `bondbreak` is used in this thesis for simulating the diffusion. With `bondbreak` local distortions used to calculate η are based on distance between atoms. With `angleswitch` local distortions are instead based on dihedral angles inside quadruples of atoms.

4.3.1 `bondBreak` class

The main functionality of `bondbreak` CV is implemented in a source file `colvarcomp_bondbreak.cpp` as a class `colvar::bondbreak`. The constructor `colvar::bondbreak` is called if inside `colvar` block in the configuration file for the

COLVARS module there is a sub-block named `bondBreak`. An example of the `colvar` block containing `bondBreak` sub-block is given below

```
colvar {
  name coord
  width 0.01               # CV width, default = 1.0

  bondBreak {
    group1 {
      atomNumbersRange 1 - 721    # Group of first bond partners
    }
    group2 {
      atomNumbersRange 1 - 721    # Group of second bond partners,
                                  # can be same as first
    }
    rmin 2.57531780694319  # rmin
    rmax 3.30              # rmax
    rcut 3.00              # rcut
    waitTime 2500          # t_w
    power 8                # p
  }
}
```

Keywords `name` and `width` are general `colvar` keywords. `Name` is used to identify this CV later in COLVARS configuration file. `Width` is used for those biasing and analysis methods and CVs where bins or grids are used or for scaling if several CVs are combined. `Width` should generally be no larger than the standard deviation of the CV in an unbiased simulation. Values for `bondbreak` CV lies naturally in a range [0,1] and thus value 0.01 is better suited for CVHD than the default value of 1.0. In the context of dCVHD `width` is used to scale value of w_k in equation 24.

Keywords `group1` and `group2` specifies atoms between which length of bonds are calculated for local distortions, χ_i. Atom groups can be specified using syntax described in COLVARS manual (Bernardin, et al. 2020), in this example both groups consist of atoms having id between 1 - 721 in LAMMPS, but it is also possible to specify `group1` and `group2` so that they are not identical, but contain different atoms. Atom groups are stored in the member variables of base class of `colvar::bondbreak` as type `cvm::atom_group`. As a general rule, the size of atom groups should be kept relatively small (up to a few thousands of atoms, depending on the size of the entire system in comparison).

Keyword `power` refers to parameter p in equation 19. Keyword `waitTime` is the number of timesteps after which the system is assumed to have undergone a transition if η remain equal to 1

during all those timesteps. `waitTime` is also a number of timesteps the system equilibrates after the transition before bias force is started to apply again. This is parameter t_w in chapter 3.2. The exact logic regarding `waitTime` is explained later in this chapter.

Keywords `rmin` and `rmax` refer to parameters with the same name in equation 18. Units of `rmin` and `rmax` are Ångströms in configuration file, they are converted to the internal units of the simulation in the constructor of `colvar::bondbreak`. When simulating lattice material, `rmin` and `rmax` relate to lattice constant.

A suitable value for `rmin` is the distance to the nearest neighbor in the lattice. In the context of this thesis, the simulated material is copper which has the fcc (face-centered-cubic) lattice structure. In fcc the distance to the nearest neighbor is the lattice constant divided by $\sqrt{2}$, which can be used as a value for keyword `rmin`. It is also possible to specify a negative value for `rmin`. Then the code will calculate `rmin` as an average during `waitTime` timesteps. The average is calculated separately for each atom pair leading to individual `rmin` per atom pair when a positive value of keyword `rmin` lead to same `rmin` for each atom pair.

`rmax` should reflect the distance after which a bond between atoms is most likely broken. In the case of adatom diffusion at 001 fcc surface, the locations where the adatom can relax are located at distance of lattice constant with each other. Therefore, a working value for keyword `rmax` is somewhat less than lattice constant. Depending on a temperature the lattice constant for copper is around 3.6 Å. Value 3.3 Å, as in an example above, is thus suitable value for `rmax`. If `rmax` would be too close to the lattice constant, then χ_i could differ from 1 in the transition region and CVHD could apply bias force and break the requirement of hyperdynamics that biasing force is zero on transition region. If an adatom has moved the distance of the lattice constant, then the adatom is relaxing to a new location and transition has clearly happened. Value 3.3 Å has enough security margin for the lattice constant.

Keyword `rcut` defines the minimum distance between atoms that are counted as atom pairs in equation 19, `rcut` affect thus on parameter N of equation 19. Unit of `rcut` is also Ångström like `rmin` and `rmax`. Suitable values for `rcut` are [`rmin`, `rmax`], if `rcut` is missing from the configuration or is negative then value of `rmax` is used as `rcut`.

The main internal data structure of `bondbreak` class is a group of 3 vectors; `pairlist1`, `pairlist2` and `pairlist3`. These vectors contain atom pairs for which χ_i is calculated. `Pairlist1` contains the index of the first atom in the atom pair, `pairlist2` contains the index of the second atom in the atom pair and `pairlist3` contains `rmin` for this atom pair. The atom pair i consists of atoms having ids `pairlist1[i]` and `pairlist2[i]`. This bondlist, which is constructed from `pairlist` vectors, is totally independent concept of neighbor list, which is an essential part of MD implementation and relates to applying potential energy function to simulated atoms. The neighbor list is maintained in the LAMMPS core modules and it is not used by any CVHD module. Parameters `rmin`, `rmax` and `rcut` are not used for the neighbor list, and, respectively, neighbor list related parameters are not used by CVHD.

The main logic for calculating η of equation 21 is in function `colvar::bondbreak::calc_value()`. As Explained earlier, η varies naturally in range [0,1] and in order to be able to describe multiple consecutive events the biasing algorithm needs to reset ΔV if η remains equal to 1 during `waitTime` timesteps. To achieve this the `bondbreak` class uses variable `offset` to count how many times ΔV has reset meaning also how many transitions the system has undergone. In the context of this thesis `offset` counts number of adatom diffusion events. The variable `offset` is initialized to zero in the constructor of `bondbreak` class. The CV reported by this CVHD implementation is a combination of `offset` and η.

The function `colvar::bondbreak::calc_value()` works in 3 different states; *cleanlist*, *calculate* and *transition*.

In the cleanlist state, the algorithm loops through atom pairs in `pairlist` variables and removes all pairs where mutual distance is greater than specified by the keyword `rcut`. This is done for `waitTime` timesteps. The atom pair is removed from the list of atom pairs if the mutual distance of atoms is greater than `rcut` even once during `waitTime` timesteps. After `waitTime` timesteps the algorithm moves to state calculate. In this state a fictious value of -0.5 is used for η.

In calculate state, the algorithm simply follows the equation 21. The algorithm loops through all atom pairs in `pairlist` variables, defines mutual distances between atoms in atom pair, calculates local distortion χ_i and combines local distortions to η. When defining mutual distances

between atoms the algorithm doesn't need to take periodic boundary conditions into account because core LAMMPS and COLVARS module are taking care of them. The algorithm moves from calculate state to transition state when η is 1, or to be exact when $\eta > 0.9999$. Because of computational safety floating point number η is not directly compared to 1 but a small tolerance is allowed. During calculate state η has values [0,1[.

In transition state, η is almost equal to 1 and the algorithm follows if η has this value for `waitTime` timesteps. If η drops smaller than 1 (− the allowed computational safety tolerance) then the algorithm resumes back to calculate state and it is assumed that transition has not happened. This corresponds to a situation where an atom is in the transition region but falls back to the original location just before going over the potential barrier. If, on the other hand, η has a value close enough to 1 for `waitTime` timesteps, then it is assumed that a transition has happened. In the context of this thesis, the mutual distance of atoms in atom pair has increased longer than `rmax` Å. This means most likely adatom diffusion event because atoms in lattice are very unlikely to change places. When the algorithm finds that a transition has happened, then variable `offset` is increased by one, the list of atom pairs having mutual distance less than `rcut` is regenerated and the algorithm moves to state cleanlist. The regeneration of the list of atom pairs means that `pairlist` vector variables are emptied and mutual distances between all atoms are run through and those atom pairs where mutual distance is shorter than `rcut` are added to `pairlist` variables. During transition state η is 1 (or very close to 1) all the time.

As mentioned, the actual CV reported by `bondbreak` class is a combination of `offset` and η. The formula to for reported CV is

$$CV = 2 * \text{offset} + \eta \tag{32}$$

In the beginning of the simulation, `offset` is 0 and the algorithm is in the cleanlist state, thus $\eta = -0.5$, which results to a CV of -0.5. When the algorithm moves first time to calculate state then CV = [0,1[and when the algorithm is first time in the transition state, then CV = 1. After the first transition `offset` = 1 and in cleanlist state CV = 1.5, in calculate state [2,3[and in transition state 3. Thus CV increases by 2 after every transition and CV values of even number minus 0.5 (-0.5, 1.5, 3.5, …) indicate cleanlist state, CV values between even and odd number ([0,1[, [2,3[, [4,5[, …) indicate calculate state and CV values of odd number (1, 3, 5, …) indicate transition state. It is

possible to obtain `offset` and η from CV using `floor` function which returns the nearest integer which is not greater than the argument of `floor` function.

$$\text{offset} = \text{floor}\left(\frac{CV + 0.5}{2}\right) \tag{33}$$

$$\eta = \text{CV} - (2 * \text{offset}) \tag{34}$$

By knowing CV, it is thus possible to know how many transitions the system has undergone, this is directly value of `offset`, and in which state the algorithm is. This way CVHD provides different range of values for each transition and for states preceding this transition. In this sense CV in CVHD differentiates relevant states of system as required by metadynamics (Fiorin & Klein & Hénin 2013). CV values related to the first transition and states before that are in range [-0.5, 1], for second transition [1.5, 3] and so on. However, as earlier mentioned, η doesn't fully differentiate all possible combinations of atom locations and it might be that same value of η is obtained for two different states of the system, which leads to same value of CV for two different states. The same value of CV, for two different states, is however not a problem, because in CVHD interest is in transitions and for each transition CV is different.

4.3.2 `colvarbias_cvhd` class

The main functionality of applying biased CVHD potential is implemented in a source file `colvarbias_cvhd.cpp` as a class `colvarbias_cvhd`. The constructor `colvarbias_cvhd::colvarbias_cvhd` and initialization function `colvarbias_cvhd::init` is called if in the configuration file there exists a block named `cvhd`. The configuration for `colvarbias_cvhd` contains some common settings and some settings related the variation of CVHD (sCVHD or dCVHD) to be applied. An example of the common settings for `cvhd` sub-block is given below

```
cvhd {
  name hd                  # Something descriptive (or not)
  outputEnergy on          # Communicate energy to calling program
  colvars coord            # The CV (ONLY ONE ALLOWED!)
```

All these keywords are general keywords for the biasing and analysis method of Colvars module. Keyword `name` defines the name of the cvhd instance. The keyword `outputEnergy` defines if

the current value of the biasing energy will be written to the trajectory file, which is described in chapter 4.3.4, during the simulation.

Keyword `colvars` selects by name all the CV variables to which this bias or analysis will be applied, for `colvarbias_cvhd` only one CV is allowed. The function `colvarbias_cvhd::init` returns error if multiple CVs are tried to be defined. This example relates to the example in chapter 4.3.1 where a `colvar` named `coord` is defined. The reference to the CV object defined by `keyword` colvar is provided for `colvarbias_cvhd` class by base classes in 0^{th} element of array named `colvars; colvars[0]`.

The biasing energy is calculated in a function `colvarbias_cvhd::update`. This function updates variables `bias_energy` and `colvar_forces[0].real_value` which are used to communicate biasing potential and force to other parts of COLVARS module. `bias_energy` is defined in base class `colvarbias` of `colvarbias_cvhd` class. `colvar_forces[0].real_value` refers to member variable of CV object defined by keyword `colvars`. First this function defines the current value of CV by calling `colvars[0]->value()` and divides CV to `offset` and η using equations 33 and 34. Then variables `bias_energy` and `colvar_forces[0].real_value` are zeroed. After that bias energy and force is calculated based on variation of CVHD to be used and updated to `bias_energy` and `colvar_forces[0].real_value.`

For static CVHD only two keywords are needed

```
dynamicBias off          # Toggle dynamic/static bias
maxBias 0.3              # Maximal bias in the static case
```

The value `off` for the keyword `dynamicBias` tells `colvarbias_cvhd` class to use static cvhd (sCVHD). The keyword `maxBias` correspond to parameter ΔV^{max} in equation 22. Units of `maxBias` are units of energy in MD simulation. As mentioned earlier, `maxBias` should be comparable to the natural energy barriers in the system. In the context of this thesis, when a metal is simulated, units of `maxBias` are electron volts. With sCVHD `colvarbias_cvhd::update` checks if $\eta < 1$ and if yes `bias_energy` is set according to equation 22 and `colvar_forces[0].real_value` is added by ΔV^{max}, that is, added by a value specified by

keyword `maxBias`. If $\eta \geq 1$ then `bias_energy` and `colvar_forces[0].real_value` are left to zero. sCVHD is not used in this thesis.

For dynamic CVHD the following keywords can be used

```
dynamicBias on            # Toggle dynamic/static bias
hillWidth 1.0             # Width of Gaussian
hillWeight 0.005          # Height of Gaussian
newHillFrequency 1000     # Frequency of Gaussian
```

The value `on` for the keyword `dynamicBias` tells `colvarbias_cvhd` class to use dynamic cvhd (dCVHD). For dCVHD `colvarbias_cvhd` class has member variable `hills`, which is a vector containing gaussian hills that are added as biasing potentials. Keywords `hillWidth`, `hillWeight` and `newHillFrequency` correspond to parameters σ, w_k and τ_G in equation 24. σ is obtained by multiplying value of keyword `hillWidth` with value of the keyword `width` in the block `colvar`. Units of `hillWidth` and `hillWeight` correspond to the units of CV, in case of `colvar::bondbreak` units are Ångtroms. The unit of `newHillFrequency` is number of timesteps.

With dCVHD the function `colvarbias_cvhd::update` first divides the value of CV (obtained from `colvars[0]->value()`) to `offset` and η according to equations 33 and 34. If `offset` has changed, then vector `hills` is cleared, and this way bias energy will be zeroed after the transition.

Next the function `colvarbias_cvhd::update` checks if a new hill needs to be added on vector `hills`. The new hill is added if `newHillFrequency` timesteps has elapsed since the last addition of a hill and if $\eta < 0.9$. Upper limit of η for adding a new hill restrict adding bias on large η values. For the new hill following values are stored in vector hills:

- Center, the parameter $\eta(k\tau_g)$ in equation 24. The current value of CV obtained from `colvars[0]->value()`,
- Weight, the parameter w_k in equation 24. This is specified by `hillWeight`,
- Width, the parameter σ in equation 24. This is obtained by multiplying value of keyword `hillWidth` with value of the keyword `width` in the block `colvar`.

Regardless of if the new hill is added or not, the function `colvarbias_cvhd::update` updates next the variable `bias_energy`. This is done by looping through all hills stored in vector `hills` and applying equation 24 on all of them. The part $\eta - \eta(k\tau_g)$ of equation 24 is obtained by decreasing the stored value of the center of the current hill from the current value of CV. Other parameters of equation 24 are obtained from stored values of weight and width of the current hill. If, for some hill, the exponential part of equation 24 is less than -23 then the value of this hill is considered as zero in order to avoid computationally too small numbers. This loop accomplishes the sum in equation 24. This total sum over all stored hills is then set as `bias_energy`.

The same kind of loop over the `hills` vector, as for `bias_energy` is run in order to update `colvar_forces[0].real_value`, the biasing force, except that instead of calculating $\Delta V(\eta)$ from equation 24 a negative gradient of it is calculated for each stored hill. The terms for individual hill k are format

$$f_k = w_k \frac{\eta - \eta(k\tau_g)}{2\sigma^2} \Delta V_k(\eta) \tag{35}$$

Where $\Delta V_k(\eta)$ is a contribution of k^{th} hill in the sum of equation 24. $\Delta V_k(\eta)$ is already calculated for each hill when updating `bias_energy`. This way it is not necessary to calculate $\Delta V_k(\eta)$ again in this phase. The loop summarizes $f_k$values of all hills together in the variable `colvar_forces[0].real_value`.

It is possible to modify dCVHD to use well-tempered metadynamics (Barducci & Bussi & Parrinello 2008). In well-tempered metadynamics, parameter w_k of the equation 24 is modified with history-dependent potential and parameter ΔT. Well-tempered metadynamics increases the barrier crossing and thus rate of the transitions and facilitates the more efficient exploration in the CVs space than standard metadynamics. In standard metadynamics, values of η that are frequently visited will be disfavored in the long run, because many bias hills are stored near those values of η, well-tempered metadynamics overcomes this limitation by controlling weight of the hills, w_k. In this implementation of dCVHD the metadynamics effect is defined as

$$w_{k+1} = w_0 \exp\left(-\frac{\Delta V(\eta)}{k_B \Delta T}\right) \tag{36}$$

where w_{k+1} is the weight of the new hill added by the function `colvarbias_cvhd::update`, w_0 is the "base" weight of a hill defined by keyword `hillWeight`, $\Delta V(\eta)$ is the bias potential cumulated according to the equation 24 before adding a new hill, and ΔT is scaling parameter in units of Kelvin. It should be noted that ΔT has no physical interpretation, it is a parameter, telling how much normal metadynamics algorithm is modified. On the limit $\Delta T \to 0$ also $w_{k+1} \to 0$, no hills are added and well-tempered metadynamics return to standard MD. On the other hand, at limit $\Delta T \to \infty$ the exponent function approach to 1 leading to $w_{k+1} = w_0$, which is standard metadynamics. (Barducci & Bussi & Parrinello 2008).

To use well-tempered metadynamics with dCVHD, the following keywords are needed

```
wellTempered on           # Use Well-Tempered Metadynamics
biasTemperature 2000      # Well-Tempered Metadynamics bias temperature
```

The keyword `wellTempered` with value `on` turns on well-tempered metadynamics and keyword `biasTemperature` corresponds to parameter ΔT in equation 36. These keywords modifies the execution of the function `colvarbias_cvhd::update` so that before adding a new hill, value of $\Delta V(\eta)$ with existing hills is calculated, then equation 36 is applied and a new hill with weight w_{k+1} is added. Finally, $\Delta V(\eta)$ including also the new hill is calculated and `bias_energy` and `colvar_forces[0].real_value` are updated like without well-tempered metadynamics.

With keyword `adaptiveEta` it is possible to modify the function `colvarbias_cvhd::update` so that the upper limit of η for adding a new hill is not fixed to 0.9, but varies. With `adaptiveEta` a new hill is added after transition only if $\eta < 0.1$ after that the limit for adding new hills is gradually increased when η grows. This way hills are first deposited on low values of η, far from the transition state and the hills for larger values of η are added later. The keyword `adaptiveEta` has values `on/off` and missing keyword corresponds setting this keyword to `off`. `adaptiveEta` feature was not used in this thesis.

4.3.3 `FixTimeboost` class

The logic for calculating hypertime is implemented in a source file `fix_timeboost.cpp` as a class `FixTimeboost`. This fix has 5 parameters

```
fix             boost all timeboost 550 cvfix
```

Like with all fix commands in LAMMPS, also with the `timeboost` fix, the first parameter is the name of the fix, the second parameter is the name of the atoms group for which the fix is applied and the third parameter is the type of the fix, `timeboost` in this case. `timeboost` ignores the second parameter of `fix` command but for the sake of clarity, it is good to use built-in group `all` as a second parameter. The fourth parameter is simulation temperature, parameter T in equation 31, this is a temperature in canonical NPT ensemble and constant throughout the simulation, not calculated from kinetic energy. The fifth parameter is the name of `colvars` fix.

The logic of the `FixTimeboost` class is quite simple. The upper classes provides variables `update` and `modify` that are used to access other parts of the LAMMPS simulation engine. With the `modify` variable `FixTimeboost` accesses `colvars` fix named by the fifth parameter and requests energy related to that fix. This energy is then used as parameter $\Delta V(\eta)$ in the equation 31. After each timestep `FixTimeboost` calculates the boost according to the equation 31.

The current simulation time is hold in variable `update->atime`. After boost is calculated this time is increased by a following line of code

```
update->atime += (boost-1.0)*dt;
```

`dt` in a code above is simulation timestep of LAMMPS. In this thesis value of a 1 fs was used as a size of timestep in all simulations. `FixTimeboost` obtains `dt` through the variable `update`. Hypertime elapsed during one timestep is `boost*dt`, but because LAMMPS updates `update>atime` by `dt` only the part of hypertime, which exceeds `dt,` is added to `update->atime`. As a result, `update->atime` contains total hypertime.

It is possible to output `update->atime` to LAMMPS output file using `thermo_style custom` command and keyword `time` in LAMMPS configuration file. If the fix `timeboost` is used, then output contains hypertime, otherwise output contains normal simulation time which is number of timesteps times dt. In this thesis metal units are used, and with metal units time is expressed in picoseconds.

4.3.4 Colvars trajectory file

If the keyword `outputEnergy` has value `on`, then the current value of the biasing energy will be written to the colvars trajectory file. Keyword `outputEnergy` is defined in COLVARS configuration file and the name of the colvars trajectory file can be defined by optional parameters of colvars `fix` command in LAMMPS configuration file. They layout of colvars trajectory file is simple, it has 3 columns; number of simulation step, value of CV (with the name of CV as header) and biasing energy. Following is a sample of typical colvars trajectory file from CVHD simulation at 350 K with one adatom at the top of copper slab. This sample shows the first transition in the simulation.

```
# step         coord                  E_hd
. . . . . . . . .
     1591500    4.17027906024275e-01   5.65883923415798e-02
     1592000    7.23314331677679e-01   1.33706348223600e-01
     1592500    7.13483925793328e-01   1.21748505975718e-01
     1593000    7.83538212985108e-01   9.85693530710990e-02
     1593500    7.93162237137970e-01   9.36349683844025e-02
     1594000    1.00000000000000e+00   0.00000000000000e+00
     1594500    1.00000000000000e+00   0.00000000000000e+00
     1595000    9.82914545805961e-01   0.00000000000000e+00
     1595500    4.72200114042248e-01   1.11189589205285e-01
     1596000    3.95400106620365e-01   3.27248661065285e-02
     1596500    6.90362015628532e-01   1.26960043362092e-01
     1597000    8.96935738860761e-01   6.33254388274207e-02
     1597500    1.00000000000000e+00   0.00000000000000e+00
     1598000    9.87559068481485e-01   0.00000000000000e+00
     1598500    7.99613305576896e-01   9.29695230523765e-02
     1599000    1.00000000000000e+00   0.00000000000000e+00
     1599500    1.00000000000000e+00   0.00000000000000e+00
     1600000    1.00000000000000e+00   0.00000000000000e+00
     1600500    1.00000000000000e+00   0.00000000000000e+00
     1601000    1.00000000000000e+00   0.00000000000000e+00
     1601500    1.50000000000000e+00   0.00000000000000e+00
     1602000    1.50000000000000e+00   0.00000000000000e+00
     1602500    1.50000000000000e+00   0.00000000000000e+00
     1603000    1.50000000000000e+00   0.00000000000000e+00
     1603500    1.50000000000000e+00   0.00000000000000e+00
     1604000    2.63250933724711e+00   5.00000000000000e-03
     1604500    2.53838094877505e+00   0.00000000000000e+00
     1605000    2.68891798334518e+00   5.00000000000000e-03
     1605500    2.70378432944920e+00   6.01612382310323e-05
     1606000    2.57509525364900e+00   5.00000000000000e-03
     1606500    2.70054290811794e+00   3.35095535568371e-04
     1607000    2.58064350811467e+00   7.62365363459779e-03
```

In the sample above, the value of CV named `coord` varies between [0, 1[during simulation steps 1591500 – 1593500. Based on equation 33, this value of CV tells that `offset` = 0 and no transition has happened yet. `Bondbreak` algorithm is during these steps in calculate state and bias energy is applied as seen on column `E_hd`. During simulation steps 1594000 – 1594500 CV has value 1 and `bondbreak` algorithm is in transition state waiting to see if there is a persistent transition. In this simulation keyword `waitTime` had value 2500 and because CV returned to a value below 1 before 2500 steps has been elapsed, the `bondbreak` algorithm returned to the calculate state. It can be seen from the column `E_hd` that bias is not deposited while the algorithm is in transition state.

The same alternation between calculate and transition state happens during steps 1595000 – 1598500. At step 1599000 the simulation enters once again transition state, CV is 1 and `E_hd` = 0. This time the transition is persistent, CV remains on value 1 for 2500 steps, until step 1601000. Then the `bondbreak` algorithm increases `offset` from 0 to 1 and moves to cleanlist state. In the cleanlist state the reported CV is 2 x `offset` – 0.5 which is 1.5 when `offset` = 1. This value is seen from step 1601500 to 1603500 when the `bondbreak` algorithm checks for 2500 step which bonds in bondlist are persistent. Also, during this time `E_hd` = 0. At step 1604000 the `bondbreak` algorithm moves to the calculate state with `offset` = 1 and thus reported CV values are in a range [2, 3[and `E_hd` > 0 telling that bias potential is deposited. The occurrence of the next transition can be seen in the colvars trajectory file when CV stays in value 3 for 2500 timesteps.

This is a very typical output of CVHD simulation. The `bondbreak` algorithm moves between calculate and transition states when the simulation approaches transition range but doesn’t quite exceed the potential barrier between two different simulation states and falls back to the previous simulation state. After several attempts, the transition occurs and the `bondbreak` algorithm stays in transition state for `waitTime` simulation steps. Then, after cleanlist state, an evolution towards next transition starts.

In this sample, CV (`coord`) and `E_hd` were reported every 500 steps. Thus, the description above is inaccurate in a sense that the change of state of the `bondbreak` algorithm doesn’t occur exactly at the step shown in the colvars trajectory file. For example, when changing from state calculate to

the state transition, the step 1594000 is the first step divisible by 500 after the state change. The state could have been changed in any step between 1593501 and 1594000.

The frequency of reporting CV and `E_hd` can be controlled by keyword `colvarsTrajFrequency` in COLVARS configuration file. `colvarsTrajFrequency` is located outside of any blocks in COLVARS configuration file. If the value is 0, then Colvars trajectory file is not written. Otherwise CV values are written to the file by intervals specified by this keyword. If one is interested in following how the `bondbreak` algorithm works, then `colvarsTrajFrequency` should be less than `waitTime`. For other purposes `colvarsTrajFrequency` can be larger resulting in smaller COLVARS trajectory file.

4.3.5 Communicating biasing force to atoms

All classes and algorithms are tied together when a biasing force is communicated to atoms. As an example, we study how biasing force is applied to atom with id 721 during simulation step 5000 in a surface diffusion simulation at 300K. From colvars trajectory file CV and `E_hd` for this step can be seen.

```
5000     4.09724896084701e-01    5.04451401836426e-03
```

The first CVHD related action during this simulation step is when COLVARS module calls the function `colvar::bondbreak::calc_value()`, which calculates the value of CV 0.409724896084701.

Then COLVARS module calls the function `colvar::bondbreak::calc_gradients()`, which calculates and stores the gradient for each atoms. First this function checks if $\eta < 1.0$ and if not, then heavy calculation, related to gradients, is not done because the simulation is in transition state and no biasing potential or force is applied. If $\eta < 1.0$ then function calculates first that part of equation 26, which is same for each bond

$$\pi\chi_T \sin(\pi\chi_T^2)\left(\sum_i^N \chi_i^p\right)^{1/p-1} \tag{37}$$

Then `colvar::bondbreak::calc_gradients()` calls auxiliary `switching_function<true>` function for each bond. `switching_function<true>` calculates bond specific part of equation 26

$$\frac{\left(\frac{r_i - r_i^{min}}{r_i^{max} - r_i^{min}}\right)^{p-1}}{r_i^{max} - r_i^{min}} \hat{\boldsymbol{r}}_{\boldsymbol{i}} \tag{38}$$

The total gradient for one bond is obtained by multiplying equations 37 and 38 together. COLVARS module implements vector arithmetics so the gradient can be manipulated as 3-dimensional vector. All variables needed in equations 37 and 38 are available in the class `colvar::bondbreak` and its bondlist variables. The gradient is added to `grad` member variable of both atoms of bond *i* but with opposite signs, because the force along the bond goes on opposite direction when looking from opposite ends of the bond. The gradient is added to `grad` member variable because the same atom might exist in multiple bonds and by adding the effect of each bond the total gradient for one atom is obtained. The atom-specific gradient stored in `grad` member variable is used later during this simulation step. In this case gradient for the atom with id 721 is (-8.10841e-10, 9.83435e-09, 1.08626e-08).

Next COLVARS module calls all biasing and analysis methods defined for the simulation. In this case, only the CVHD biasing method is defined and COLVARS module calls the function `colvarbias_cvhd::update` which calculates biasing potential 0.00504451 and force -0.0280868 as explained earlier. The force, that `colvarbias_cvhd::update` stores on `colvar_forces[0].real_value`, is passed to member variable `fb` of the class `colvar` through several steps; `colvarmodule::update_colvar_forces()`, `colvarbias::communicate_forces()` and `colvar::add_bias_force`. The class `colvar` is one of base classes of `colvar::bondbreak` and thus the member variable `fb` is accessible in the `colvar::bondbreak` class. This is a general method how COLVARS module communicates the force from a biasing method to CV.

When the biasing force is communicated from the biasing method to CV, the COLVARS module directs CV to communicate forces to atoms belonging to CV. This process starts in `colvarmodule::update_colvar_forces()`, which calls

`colvar::communicate_forces()` and which calls `colvar::bondbreak::apply_force`. `colvar::bondbreak::apply_force` then calls `cvm::atom_group::apply_colvar_force` for atom group in member variable `group1` and calls the same function also for member variable `group2` if it differs from `group1`. In the function `cvm::atom_group::apply_colvar_force` the next function, `colvarmodule::atom::apply_force`, is called for each atom in a group with value biasing force multiplied by gradient. The gradient is earlier stored in member variable `grad` of atom class and is utilized here. This is the step where the biasing force calculated by `colvarbias_cvhd class` and the gradient calculated by `colvar::bondbreak` class are combined and applied to atom level. In this example the applied force is (2.2774e-11, -2.76216e-10, -3.05097e-10).

`colvarmodule::atom::apply_force` finalizes the job by calling `colvarproxy_atoms::apply_atom_force`, which simply adds the new force to an atom-specific array of forces. From this array biasing forces are communicated to the main LAMMPS simulation engine to be added to forces derived from the potential energy functional used in the simulation.

This process of passing biasing force from CVHD algorithm to individual atoms is done in every simulation step. The process is quite complex but, the reason for this complexity is that there are several flags and options how the force is communicated to atoms in detail. The process shown here is only one option and adjusted for CVHD. With different CV or with different biasing/analysis method, the COLVARS module can be fine-tuned to use force communication process specific to the current setup.

4.3.6 Programmatic considerations

In order to include CVHD in LAMMPS, the CVHD related new source files need to be added and some modifications need to be done to existing source files. Then COLVARS module need to be compiled resulting in library file which is linked with LAMMPS core. These steps are explained in more detail next.

The instructions for obtaining LAMMPS source code are available in LAMMPS website https://lammps.sandia.gov/ (Sandia National Laboratories). The LAMMPS distribution includes

source code both LAMMPS core, in `src` –directory, and source code for various additional modules in module-specific subdirectories in `lib` –directory. Source files for COLVARS module are in the directory `lib/colvars`. The `src` –directory of LAMMPS distribution also includes Makefile, which is quite versatile and highly automated tool for controlling how LAMMPS is built. All options for using Makefile of LAMMPS can be seen by writing `make help` or just `make` in command line in the `src` directory.

CVHD source files are available as a supporting material of Bal's & Nyets' article (Bal & Neyts 2015b). This CVHD package contains a directory named LAMMPS containing two files `fix_timeboost.cpp` and `fix_timeboost.h`. These two files implement FixTimeboost class described in chapter 4.3.3. Due to automated nature of Makefile it is enough just to copy these two files into the `src` –directory of LAMMPS and they will be part of compiling and linking LAMMPS.

The other essential part of the CVHD package is the directory named `colvars`. The `colvars` directory of the CVHD package contains files `colvarbias_cvhd.cpp`, `colvarbias_cvhd.h`, `colvarcomp_bondbreak.cpp`, `colvarcomp.h` and `README.txt`.

Files `colvarbias_cvhd.cpp` and `colvarbias_cvhd.h` implement the class `colvarbias_cvhd` described in chapter 4.3.2, these two files can be directly placed in the directory `lib/colvars`. Files `colvarbias_cvhd.cpp` and `colvarbias_cvhd.h` implement the class `colvar::bondbreak` described in chapter 4.3.1. The file `colvarcomp_bondbreak.cpp` can be directly placed in the directory `lib/colvars`. The class `colvar::bondbreak` has not its own header file, but the declaration of this class is in the file `colvarcomp.h` together with declarations of several other CV classes. Therefore the content of `colvarcomp.h`, which relates to `colvar::bondbreak`, need to be merged with the content of `colvarcomp.h` which comes as a part of LAMMPS distribution. The same applies when updating LAMMPS. There might be changes in other parts of `colvarcomp.h` than those related to `colvar::bondbreak` and therefore it is better to merge `colvarcomp.h` with the new instance of this file instead of copying modified `colvarcomp.h` from the older version of LAMMPS which has CVHD already implemented. Together files

colvarcomp_bondbreak.cpp and colvarcomp.h implement the class colvar::bondbreak described in chapter 4.3.1. The declaration of the class colvar::bondbreak was modified in the context of this thesis by commenting out distance functions dist2, dist2_lgrad and dist2_rgrad in order to avoid compilation errors. As a result, instead of colvar::bondbreak specific distance functions the corresponding distance functions of the base class are used.

The file readme.txt contains instructions how to modify various files in lib/colvars directory for adding CVHD as a part of COLVARS module.

Source files for CVHD addon required some modifications, because the LAMMPS version (and thus COLVARS module) has changed since 2015 when the CVHD was firstly published. Also, instructions in the file readme.txt requires modifications.

In addition to adding new source files in lib/colvars directory and embedding the declaration of the class colvar::bondbreak in the file colvarcomp.h, it is necessary to embed colvar::bondbreak and colvarbias_cvhd related code snippets to various other files so that COLVARS module will compile and utilize CVHD.

In readme.txt it is instructed to add a variable in a file colvarmodule.h for counting number of activated CVHD biases. Since the publication of the CVHD package, the COLVARS module has been changed so that counting of number of instances of bias classes is not done anymore. Therefore, this modification of colvarmodule.h is not necessary anymore. For the same reason, the change to file colvarbias.cpp, instructed also in the file readme.txt, is not needed anymore.

The file colvarmodule.cpp needs to be modified by including file colvarbias_cvhd.h and adding “cvhd” as one string that is recognized when parsing the COLVARS configuration file for biasing methods. As mentioned earlier, COLVARS doesn’t count number of instances of bias classes anymore and therefore it is not necessary to add initialization of the variable containing number of colvarbias_cvhd instances in colvarmodule.cpp even if readme.txt suggest doing this. As a part of this thesis, the check for the return of possible error code from the init function of any of bias classes was added to colvarmodule.cpp.

Files `colvar.h` and `colvar.cpp` need to be modified to take into account the new bondbreak CV. The forward declaration of the class `bondbreak` needs to be added inside the declaration of the class `colvar` in the file `colvar.h`. The initialization of class `colvar::bondbreak` needs to be added inside the function `colvar::init_components` in the file `colvar.cpp`. This is the function where also other CV classes are initialized. These modifications are as described in `readme.txt`.

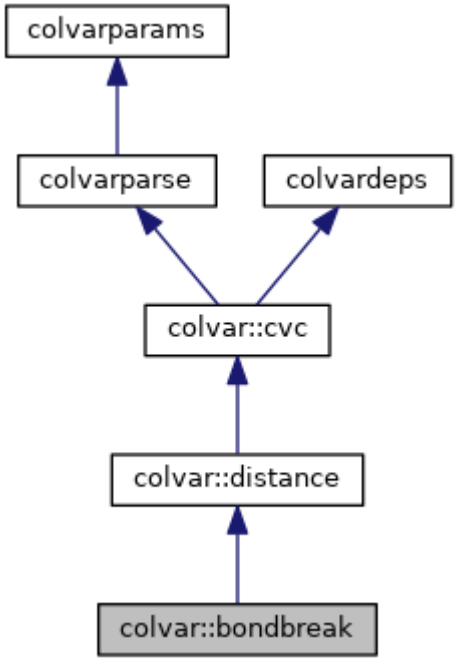

Figure 4: Class diagram of `colvar::bondbreak`

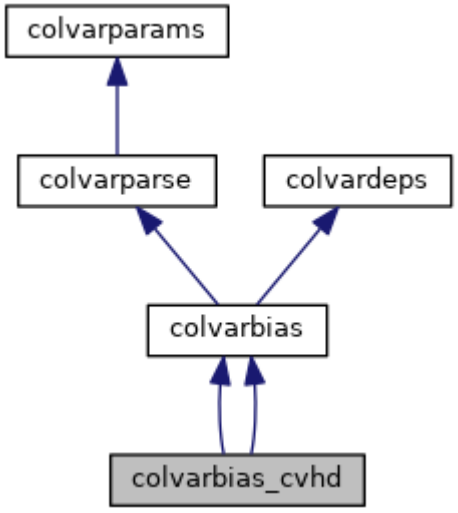

Figure 5: Class diagram of `colvarbias_cvhd`

Also the base implementation of classes `colvar::bondbreak` and `colvarbias_cvhd` requires changes in order to work with the current version of COLVARS module. Class diagrams of these two classes are presented in figures figure 4 and figure 5.

These class diagrams are generated from source files using the Doxygen tool. The class `colvardeps` contains a set of features that control how COLVARS module works and passes biasing potential and energy between different classes. The other base classes than `colvardeps` provides common functionality for CV and bias classes. In the code features provided by the `colvardeps` class are controlled using functions `enable` and `disable` like in a following sample.

```
enable(f_cvb_apply_force);
```

The class `colvardeps` has different flags for CVs (that is classed derived from `colvar:cvc`) and for biases (classes derived from `colvarbias`) as well as for other types of objects. Names of features for CVs start with `f_cv_` and for biases with `f_cvb_`. The use of `colvardeps` is one of the fundamental changes of COLVARS module between the time when the CVHD package was published and when this thesis was written. When the CVHD package was published different features were controlled by member variables of respective classes.

The feature that needs to be enabled for `colvar::bondbreak` is `f_cv_gradient` and the feature that needs to be enabled for `colvarbias_cvhd` is `f_cvb_apply_force`. The version of COLVARS, that was used when the CVHD package was published, has no options for these features, they were always set. For the current version of the COLVARS module, these features need to be actively enabled for the biasing force to be communicated to the atoms as described in chapter 4.3.5. `f_cv_gradient` instructs COLVARS module to call `colvar::bondbreak::calc_gradients()`. Without enabling `f_cvb_apply_force` the function colvarbias`::communicate_forces()` does nothing, the COLVARS module then assumes that instead of biasing method an analysis method, which doesn't generate biasing force, is used.

The other CV classes in the current implementation of COLVARS use iterators for accessing atom groups but this change was not done in the `colvar::bondbreak class` because it would have caused major re-write of code. Iterators are a more modern way to handle vectors than accessing with indices like in the published CVHD package and thus it would be feasible to modernize code from this point. But from the performance point of view, accessing vectors with iterators or indices does not make a significant difference.

Makefile logic for COLVARS module is built so that there are Makefiles with different extension for different configurations, for example `Makefile.serial` for LAMMPS without distributed memory parallelization or `Makefile.mpi` with Message Passing Interface (mpi) library for distributed memory parallelization. All these different makefiles include the file `Makefile.common`, which contains common build and link options for the different configurations of COLVARS module. `Makefile.common` in turn includes automatically built `Makefile.deps` that contains build targets of different source files. In order to include CVHD in

COLVARS build prosess it is first necessary to add filenames `colvarbias_cvhd.cpp` and `colvarcomp_bondbreak.cpp` to a variable `COLVARS_SRCS` in the file `Makefile.common`. Then existing `Makefile.deps` can be removed and the following command entered to generate a new `Makefile.deps` file including CVHD build targets.

```
make -f Makefile.common Makefile.deps
```

This is somewhat simpler way than adding CVHD build targets directly to configuration specific makefile as suggested in `readme.txt`.

The CVHD implementation used in the context of this thesis was built with mpi configuration. The MPI library used was OpenMPI, an open source MPI-3 implementation (Gabriel, et al. 2004). The used computer environment in the University of Helsinki requires first issuing `module` command to load necessary path, environment variables and other settings for OpenMPI and compiler. After that, the following command can be entered in order to actually build COLVARS module

```
make -f Makefile.mpi
```

This produces, if there are no errors, a library file named `libcolvars.a`. So far the build process has taken place in the directory `lib/colvars`. The rest of build process happens in `src` - directory of LAMMPS.

The makefile of LAMMPS core module is a versatile tool for controlling many aspects of building LAMMPS executable. The instructions on how to use LAMMPS makefile can be obtained by command `make | more`. The first task in building LAMMPS is to add necessary modules in the build process. This can be done with command `make yes-package`, where `package` is the name of module to be included in the build. In order to use CVHD the COLVARS module needs to be included with command `make yes- user-colvars`. In order to keep LAMMPS executable as light as possible, without unnecessary modules, only manybody module was included in LAMMPS build, used with this thesis, in addition to COLVARS module. The manybody module is needed for EAM potential used with this thesis.

Makefile of LAMMPS uses a concept “machine” for different target configurations. “Machine” will be given as a parameter for `make` command. The mpi “machine” was used in the context of this

thesis and so command `make mpi` was used to build LAMMPS executable. With these setting the resulting executable has name `lmp_mpi`. By typing command `lmp_mpi -h` it can be checked that all necessary components are in place. Output for installed packages should contain manybody and user-colvars, and output for fix styles should contain colvars and timeboost.

All CVHD related new and modified files are available from GitLab of the University of Helsinki (https://version.helsinki.fi/) under the project "Mika Kurki / CVHD for LAMMPS". The modifications in files are marked with string `MKu`. The files in GitLab contains also some small, technical, modifications that are not described above and that are not relevant for understanding the implementation of CVHD.

5 Setup of simulations

The physical phenomenon simulated in this thesis was adatom diffusion on the Cu(001) surface. The diffusion was simulated in different temperatures with slightly different CVHD parameters. The LAMMPS input script was almost the same for all the simulations, expect for the temperature. The simulated temperatures were between 150 K and 600 K. The simulations were run in NVT ensemble using Nose-Hoover thermostat for keeping constant temperature. In this chapter typical LAMMPS input script and COLVARS configuration file used with this thesis are presented.

The basic properties of copper that are relevant for these simulations are atomic mass 63.546(3) u, face-centered-cubic (fcc) crystal structure with lattice constant around 360 pm depending on temperature and melting point 1357.77 K. EAM potential was used for the simulations. The exact potential used was Cu_u3.eam provided as a part of standard LAMMPS distribution (Foiles & Baskes & Daw 1986).

For the simulations in this thesis metal units were used. Key metal units used by LAMMPS are

mass = grams/mole
distance = Angstroms
time = picoseconds
energy = electron volt
velocity = Angstroms/picosecond
force = eV/Ångström

temperature = Kelvin

Timestep parameter dt is 0.001 psec (= 1 fs) by default with metal units in LAMMPS.

A geometry for the diffusion simulation was a slab which had one adatom on top of it. The slab had periodic boundary conditions in x and y directions and fixed boundary conditions in z -direction. Above and below of the slab there was plenty of empty space, otherwise an atom could be lost, if it would bypass the fixed z-boundary and cause an error in the simulation. If there are no lost atoms, the slab has not too tight boundaries. This setup is specified in LAMMPS input script as follows

```
#---------- Initialize Simulation --------------------
units      metal
atom_modify     map yes    #Fix colvars requires an atom map
atom_style atomic          #default
boundary   p p f           #fixed z boundary
```

The slab was created by first creating a bulk lattice by `create_atoms` command and then enlarging the simulation box in z -direction by command `change_box` which created empty space above and below of the bulk copper. After that, an adatom was added at the top of the slab. LAMMPS handles fcc lattice so that the topmost unit cell has atoms only in the lower corner of unit cell, and in the faces of unit cell at height of the half of unit cell. Therefore, in order to maintain fcc structure the adatom above the slab, needs to be placed in one of the top corners of the unit cell. If lattice units are used with LAMMPS, this means that coordinates of the adatom need to be given as whole numbers. On the contrary, if the adatom is placed below the slab then coordinates of the adatom need to be given as half numbers because the bottom layer of atoms in the slab consist of atoms located at lower corners of the unit cell. The adatom below the slab is then located at one of the faces of the unit cell at height of the half of the unit cell.

The sample below shows how this geometry is achieved with the LAMMPS input script. In the sample fcc lattice is defined with lattice constant 3.62967287704169 for 250 K (more about lattice constants in chapter 5.2). In this sample a block of 6 x 6 x 5 unit cells is created and filled with atoms. Then z -direction of the simulation box is enlarged by 15 lattice units both up and down, so that z -coordinate is [-15, 20] in lattice units. This generates a slab that is 6 x 6 unit cells in x and y direction 5 unit cells thick and have 15 unit cells of empty space above and below the slab. The last step is to add a single adatom at top of the slab and middle in x/y -direction, that is in coordinates (3, 3, 5).

```
# ---------- Create Atoms ---------------------

lattice         fcc 3.62967287704169     # Lattice constant in 250 K
region          mycube block 0 6 0 6 0 5 # units = lattice by default
create_box 1    mycube
create_atoms    1 box

# ---------- Create Slab and add add possible atom(s) on top of it
change_box all z final -15 20
# The topmost box is only halfly filled and line goes by whole numbers
create_atoms    1 single 3 3 5
```

After defining the simulation geometry, the LAMMPS input script defines EAM potential, used in simulations related to this thesis, and other settings for integrating forces. In the context of this thesis, the most of these settings work with their default values.

```
# ---------- Define Interatomic Potential ---------------------
pair_style      eam
pair_coeff      1 1 Cu_u3.eam
## It is not necessary to specify mass, the EAM potential files list
## atomic masses. In this case 63.550 amu
## It is not necessary to specify cutoffs; they are specified in the EAM
## potential files themselves.
## In this case 4.9499999999999886e+00 Å

neighbor   1.0 bin # 2.0 Å is default for metal units
neigh_modify    every 1 delay 10 check yes
# Defaults delay = 10, every = 1, check = yes, once = no, cluster = no
```

Next the LAMMPS input script contains definitions of fixes to use. In this case `colvars` and `timeboost` fix are used as described in chapters 4.2 and 4.3.3. Also, initial velocity and Nose-Hoover thermostat are defined here. Number 250 present in fix and velocity commands is the simulation temperature 250 K used with this sample of the LAMMPS input script.

```
# ---------- Fix ---------------------
fix             cvfix all colvars colvar.LAMMPS output out250
fix             boost all timeboost 250 cvfix
velocity        all create 250 123456

fix             myfix all nvt temp 250 250 0.2
# Nose-Hoover termostat for 250 K and relaxation time of 0.2 ps
```

The final part of LAMMPS input script defines output and starts the actual simulation run. In this sample output is generated by every 1 000 simulation step with command `thermo 1000`. The command `thermo_style` defines what output will be written. The interesting point here is output

of `time` variable. If fix `timeboost` is called earlier, then `time` variable will contain hypertime and *boost* achieved by CVHD can be calculated by dividing hypertime with simulation time. If fix `timeboost` is not called, then `time` variable contains value of dt (length of timestep) multiplied by number of simulation steps executed. `Time` is expressed in time units of the current simulation, that is in picoseconds, when metal units are used. For the purposes of this thesis location of atoms was dumped to the file every 500 steps so that it would be possible to follow the trajectory of an adatom and identify adatom related hop and exchange events.

```
# ---------- Compute & Output ---------------------
thermo          1000

# Time is step x dt if fix timeboost is not called and
# hypertime if fix timeboost is used
thermo_style    custom step cpuremain time temp press etotal pe vol
f_cvfix

dump       mydump all atom 500 atom.250K.dump

# ---------- Run (or minimize) ---------------------
run                300000000
```

The COLVARS configuration file was in practice identical for all simulations run as a part of this thesis. Below is a sample of a complete COLVARS configuration file used with the same simulation in 250 K than parts of LAMMPS input scripts presented earlier. The most changing parameter in the COLVARS configuration file was the value of `rmin`. Its value was set to minimum distance between atoms in the current simulation temperature in the perfect fcc lattice, that is

$$r_{min} = \frac{a}{\sqrt{2}} \tag{39}$$

where a is a temperature dependent lattice constant. For some simulations, the parameter p was modified. The hyperdynamics part of CVHD was not used in all simulations and in these cases `cvhd` block was not present in the COLVARS configuration file. When hyperdynamics was used, it was always used utilizing well-tempered metadynamics with bias temperature of 2 000 K (Barducci & Bussi & Parrinello 2008).

```
# collective variable test: CVHD

colvarsTrajFrequency 500   # output every 500 steps
```

```
colvarsRestartFrequency 10000

colvar {
  name coord
  width 0.01               # CV width,default = 1.0

  bondBreak {
    group1 {
      atomNumbersRange 1 - 721    # Group of first bond partners
    }
    group2 {
      atomNumbersRange 1 - 721    # Group of second bond partners, can be
same as first
    }
    rmin 2.56656630484506  # rmin
    rmax 3.30              # rmax
    rcut 3.00              # rcut
    waitTime 2500          # t_w
    power 8                # p
  }
}

cvhd {
  name hd                  # Something descriptive (or not)
  colvars coord            # The CV (ONLY ONE ALLOWED!)
  outputEnergy on          # Communicate energy to calling program
  dynamicBias on           # Toggle dynamic/static bias
  hillWidth 1.0            # Width of Gaussian
  hillWeight 0.005         # Height of Gaussian
  newHillFrequency 1000    # Frequency of Guassian addition
  wellTempered on          # Use Well-Tempered Metadynamics
  biasTemperature 2000     # Well-Tempered Metadynamics bias temperature
}
```

5.1 Tools

As a part of this thesis, few tools were developed to help run and analyze simulations. These tools were developed using Perl and Python programming languages.

`submit.pl` is the main tool for running simulations. It is a Perl script which has variables for different simulation parameters. The script will be modified for a specific simulation run by setting these variables appropriately. When the script is then run it prints a summary of simulation parameters, generates the LAMMPS input script and the COLVARS configuration file, starts the actuals simulation and extracts simulation data from LAMMPS log file to a format that is easier to handle with other programs. Because `submit.pl` generates the LAMMPS input script and the

COLVARS configuration file, then only this script is needed in order to rerun some specific simulation run.

When `submit.pl` generates the LAMMPS input script and the COLVARS configuration file, it uses simulation parameters to define various parts of these input files. The simulation temperature is used to select value for lattice constant and the simulation temperature itself is substituted in many places in the input files. Parameters control if hyperdynamics part of CVHD is applied, in other words if a `cvhd` block is included in the COLVARS configuration file or not. The simulation geometry is controlled by several parameters that tell the xy -size of the slab, thickness of the slab, free space above and below of the slab and existence of top and/or bottom adatom. Using `submit.pl` help, when several simulations, that differed from each other only slightly, were run. With `submit.pl` it was enough to change the value of simulation parameter only in one place, even if the changed parameter appears in multiple places in the LAMMPS input script and the COLVARS configuration file.

A LAMMPS log file contains both simulation data printed with `thermo_style` command and textual information concerning progress of the simulation. This is difficult to process with data processing tools like Matlab or Excel. Therefore `submit.pl` calls an auxiliary Perl script `log2dat.pl` which extracts from the LAMMPS log file only data lines and their headers and then writes them to .dat file.

Data files produced by `log2dat.pl` can be further analyzed with a Perl script `runAnalyzeDatFiles.pl`. This script scans a directory for the files matching file name pattern `"*K.dat"`, counts average and standard deviation for each variable in the data file and prints averages and standard deviations in a summary file ordered by simulation temperature. The script has also on option to print figures of time evolution of the variables. The other option for the script is the number of time steps to skip before starting to calculate average and standard deviation. This can be used to skip those time steps, from the beginning of the simulation, when the simulation has not equilibrated yet.

Internally `runAnalyzeDatFiles.pl` uses one of two Python scripts `analyzeDat.py` or `analyzeLCDat.py` to calculate averages and standard deviations and to print figures. Especially printing figures is easier with Python than with Perl. The difference between `analyzeDat.py`

and `analyzeLCDat.py` is that `analyzeLCDat.py` combines the LAMMPS `thermo_style` variables relating to simulation box length (`lx`, `ly` and `lz`) into one variable for the purpose of calculating lattice constant. `analyzeLCDat.py` is used in chapter 5.2.

The Perl script `kCount.pl` is used to find adatom exchange and hop events. This script uses two sources for finding adatom events. The first source is the LAMMPS dump file which contains the locations of atoms during different simulation steps. The script first finds an atom with the highest z -coordinate and assumes this to be an adatom. Then the script follows, if the adatom moves more than 0.9 times the lattice constant from the original location or if the atom id of the atom with the highest z -coordinate changes. The second source is the COLVARS trajectory file (chapter 4.3.4). The script analyzes from the CV value in the COLVARS trajectory file in which state of the bondbreak CV algorithm (chapter 4.3.1) the simulation is. The script will notify adatom event when the change of value of CV indicates a transition, that is when CV changes from a whole odd number (for example 3.0) to an even number minus 0.5 (for example to 3.5).

In practice it was found out that `kCount.pl` doesn't identify adatom events reliable enough from the LAMMPS dump file. The problem is probably that the script doesn't take into account back and forth movement; that is the adatom might return to the original places even when it is almost in the new place. This might work better if the script would analyze few next simulation steps to see if the adatom stays in a new location after finding the possible adatom event from the LAMMPS dump file. The analysis of the COLVARS trajectory file works reasonably well as shown in chapter 6.1. The bondbreak CV algorithm has a concept of `waitTime`, which ensures that the adatom event is registered from the COLVARS trajectory file only if the transition to a new state is permanent.

The `kCount.pl` script produces three output files. One starting with the string "`step1`", the other one starting with string "`step2`" and summary file without fixed prefix. The "`step1`" file contains the adatom event analysis of the COLVARS trajectory file. The "`step2`" file contains respectively adatom event analysis of the LAMMPS dump file. The summary files combine adatom events from both sources in one file with less detail and in more readable format. It is interesting to notice that `kCount.pl` tells η to be in transition state tens, hundreds or even more than a thousand times before a permanent transition occurs.

5.2 Lattice constant

Because the lattice constant will vary according to the temperature, a series of simulation was run in order to define the lattice constant in different temperatures. Lattice constant was defined by running a simulation of a bulk of 6 x 6 x 6 unit cells of copper, that is 864 atoms of copper when each unit cell has 4 atoms. The simulation was done in NPT ensemble using Nose-Hoover thermo- and barostat with periodic boundary conditions in all directions. Lattice constant was derived from *Lx*, *Ly* and *Lz* variables of LAMMPS. *Lx*, *Ly* and *Lz* variables in LAMMPS are dimensions of the simulation box. In NPT ensemble, the size of the simulation box changes until it starts to vibrate around the equilibrium state. The lattice constant can be defined by dividing the average of *Lx*, *Ly* and *Lz* in equilibrium state by number of unit cells in each direction.

In order to define when the equilibrium state has been reached, a series of simulation runs with different temperatures from 150 K to 2 100 K were run. It was found out that well before 250 000 steps (250 psec when 1 timestep is 0.001 psec), the system has equilibrated in all the simulated temperatures with respect to potential energy, total energy and temperature. Average of pressure was between -182 (at 2 100 K) and 666 (at 1 650 K) bars depending on the temperature and there was quite large variation of pressure around the average values. The parametrized value for pressure was 1 bar. It is a known limitation of this type of simulation, that the variation of pressure is large and only the magnitude of the average of the simulated pressure is relevant. In that sense it can be said that the system has equilibrated after 250 000 steps.

The figures figure 6 and figure 7 show the development of total energy at 300K. As can be seen the value stabilize well before 250 000 steps, in fact already during the first few thousands simulation steps.

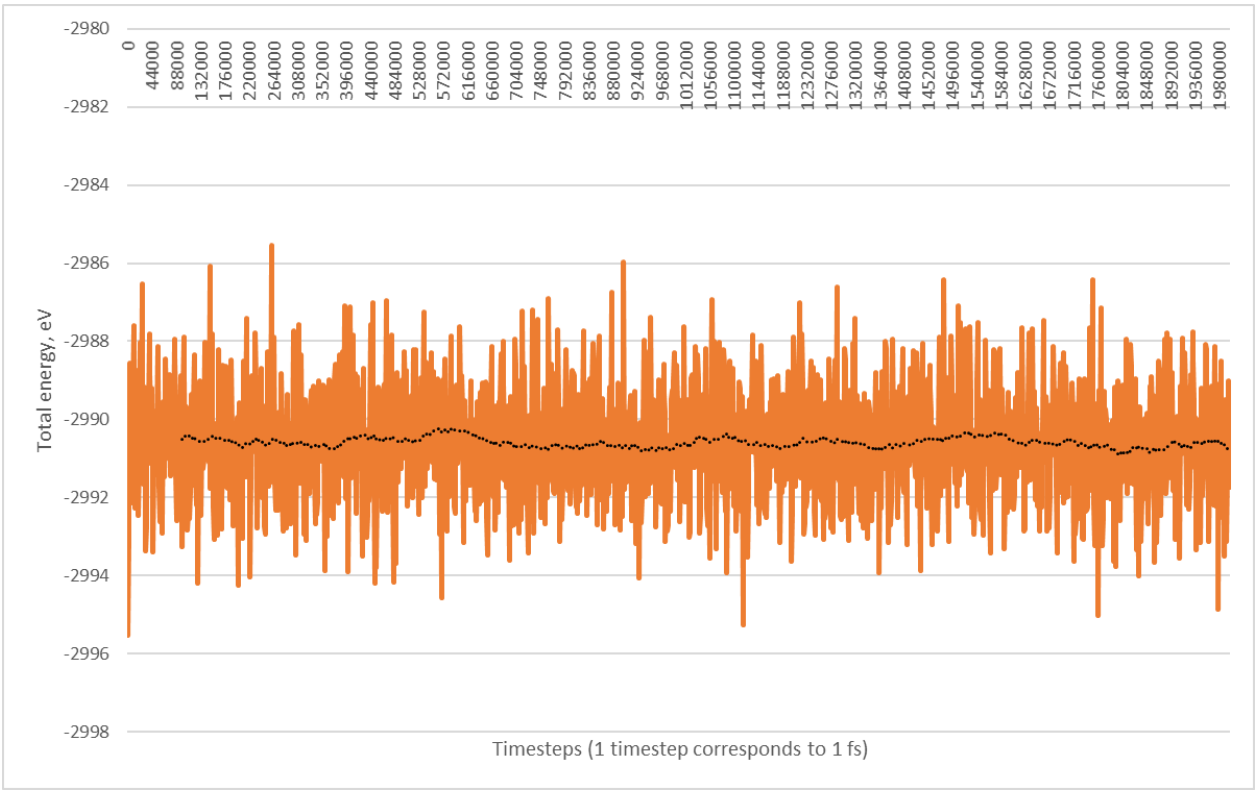

Figure 6: Total energy of the bulk simulation at 300 K. The average for total energy is -2990.61 eV and standard deviation 1,55 eV (0.052 %). The dotted, black, line is a moving average of total energy.

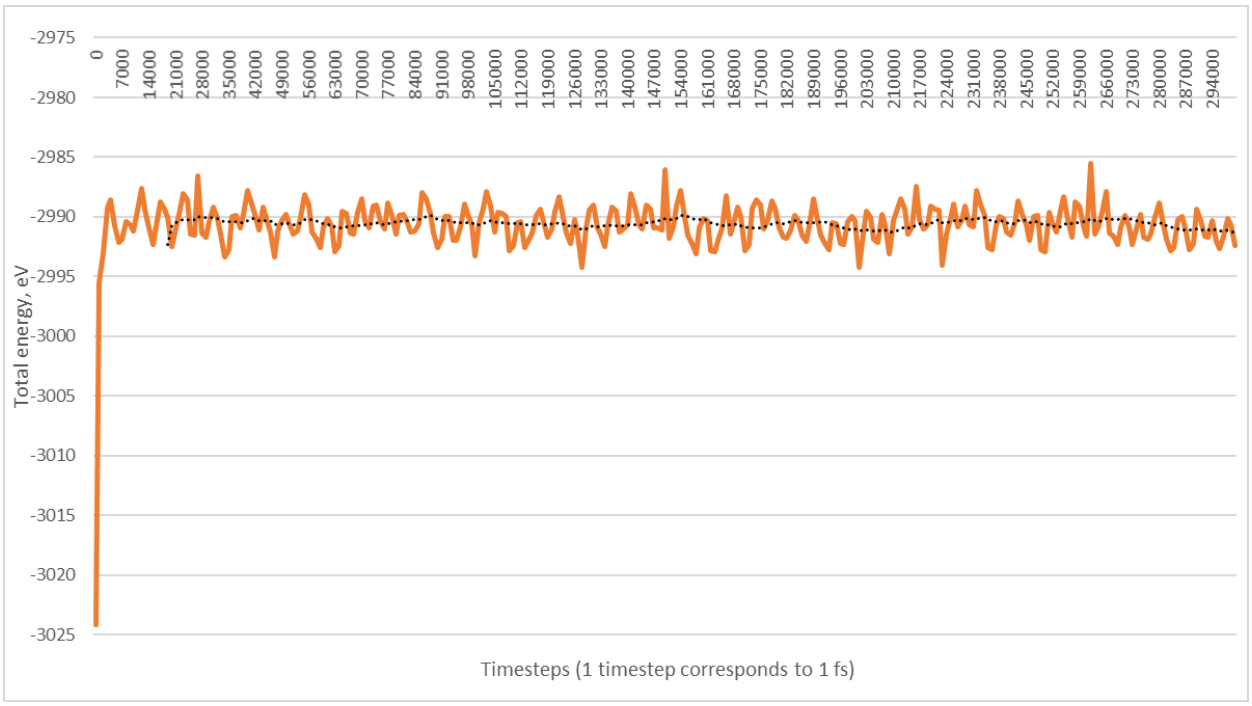

Figure 7: Total energy of the first 250 000 simulation steps of bulk simulation at 300 K. The dotted, black, line is a moving average of total energy. The value of total energy stabilizes already during the first few thousands simulation steps.

The figure for all variables (potential energy, total energy and temperature) is highly similar in all the simulated temperatures. Therefore, assuming that the system has equilibrated after 250 000 simulation steps, contains fair enough safety margin.

After the number of timesteps needed for equilibration was determined a series of bulk simulations was run in different temperatures from 150 K to 600 K with 10 K intervals. The temperature range was smaller than in equilibrium runs because the final adatom simulations were executed in the temperature range 150 K – 600 K and so lattice constants were needed only for this temperature range. The bulk simulation was run for 2 000 000 steps in each temperature and then the script `runAnalyzeDatFiles.pl` was used to calculate averages and standard deviations of lattice constant based on simulation steps from the step 250 000 to the last step of the simulation. This way the following values for lattice constants and their standard deviations were obtained.

Temp	Average of LC [Å]	Std. dev of LC	Std. dev of LC %
150	3.62376672724157	0.00149708867	0.0413 %
200	3.62672252975442	0.00175879055	0.0485 %
250	3.62967287704169	0.00195567671	0.0539 %
300	3.63269911690462	0.00363428112	0.1000 %
350	3.63571422073101	0.00398212467	0.1095 %
400	3.63892418972016	0.00416799627	0.1145 %
450	3.64204626818960	0.00453373570	0.1245 %
500	3.64529846499143	0.00476699060	0.1308 %
550	3.64858847401484	0.00504194647	0.1382 %
600	3.65190780148486	0.00522543841	0.1431 %

Table 1: Lattice constant in Ångströms in different temperatures

When these lattice constants are plotted against temperature, figure 8, it can be seen that the lattice constant increases linearly as temperature grows. This is expected because lattice constant should change according to the linear thermal expansion coefficient.

Using the least squares method, it can be estimated that the slope for the lattice constant in figure 8 is 6.25815E-05 ± 1.97385E-07. The formula for the linear thermal expansion coefficient, α, is

$$\alpha = \frac{\Delta \mathrm{L}}{\Delta T L_0} = \frac{L_1 - L_0}{(T_1 - T_0)L_0} \tag{40}$$

where L_0 and L_1 are initial and final length and T_0 and T_1 are respectively initial and final temperature. By substituting values for 150 K and 600 K from the table 1 into the equation (40) a

value of 1.72571E-05 is obtained for α. The same value is obtained also by dividing the slope (6.25815E-05) with L_0 (3.62377). This corresponds well with S. J. Bennett's experimental values where α varies from 1.696E-05 at 327 K to 1,976 E-05 at 675 K (Bennett 1978). At least in this sense the simulation and used EAM potential corresponds with experimental results.

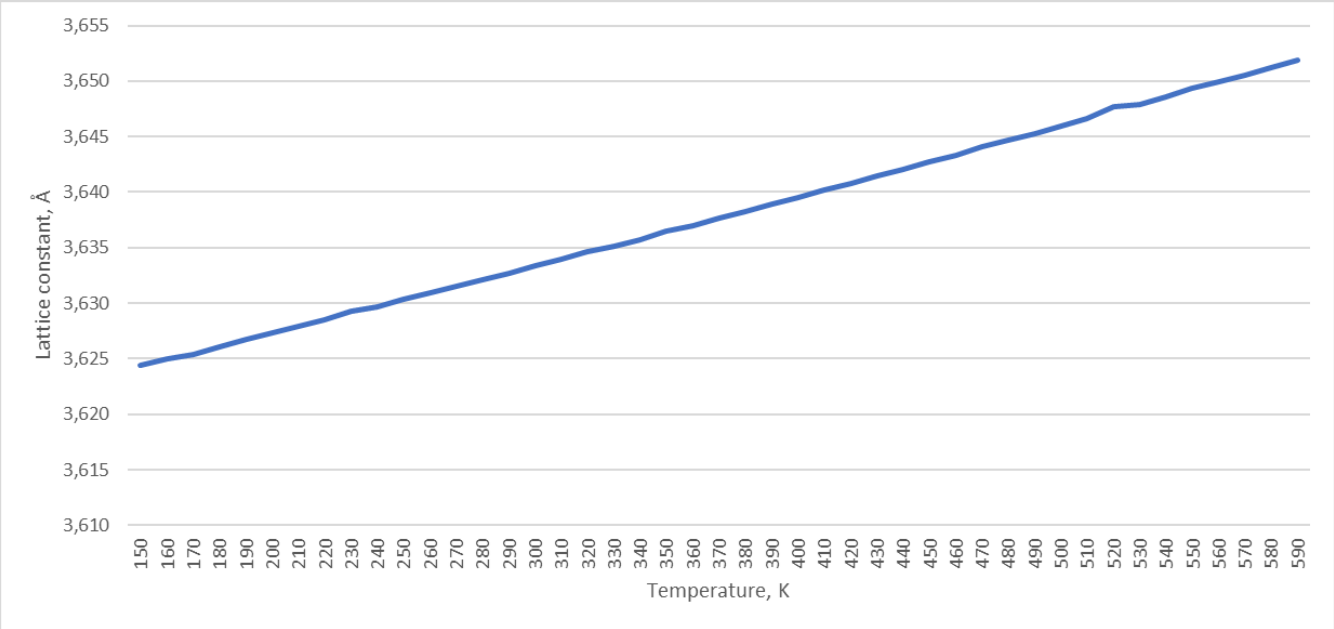

Figure 8: Value of the lattice constant (in Å) versus temperature (in K)

The lattice constants obtained were used inside the script `submit.pl` as a table from which the correct lattice constant was selected based on the parametrized simulation temperature of the current simulation run.

6 Results

Several simulations with and without CVHD was run using setup and tools described earlier. The simulations were run with varying parameters and temperatures. The parameters and result of these runs are listed in appendix A . Each run is identified with *Run Id*, so that it is easier to refer which simulation(s) are used to derive which result. Most of the initial runs were used to get the technical implementation of CVHD to work and they are not contributing to results. Only later runs are included in derivation of the results.

6.1 Accuracy of CVHD

The accuracy of CVHD method was verified in three different ways; analyzing number of adatom events indicated by bondbreak CV, analyzing Arrhenius plots and comparing deviations of system properties (temperature, total energy) during the simulations.

Run047 was used to follow the trajectory of the adatom. This run was a simulation of an adatom on top of 6 x 6 x 5 slab at 500 K for 100 000 000 simulation steps. The locations of atoms were dumped to a file after every 100 simulation step. It can be seen from the COLVARS trajectory file, that the final value of η is 26.995. So, there should be 13 adatom events. Output of `kCount.pl` script shows timesteps when the bondbreak algorithm has increased `offset`, number of observed bond breaks and how many times the simulation has been in transition state before a permanent transition.

```
   Step      CV  Offset   Progress Message
6708000    1.5       1   -0.5 Bondbreak 1, after 2498 attempts
7402500    3.5       2   -0.5 Bondbreak 2, after 215 attempts
7750000    5.5       3   -0.5 Bondbreak 3, after 113 attempts
22091000   7.5       4   -0.5 Bondbreak 4, after 4016 attempts
24388000   9.5       5   -0.5 Bondbreak 5, after 694 attempts
27912000  11.5       6   -0.5 Bondbreak 6, after 971 attempts
```

The listing above shows 6 first timesteps when `offset` has increased. The Ovito tool (Stukowski 2009) was used to visually analyze dump file around these timesteps. With visual analysis each increase of `offset` was associated with adatom events. As predicted by Evangelakis and Papanicolaou (Evangelakis & Papanicolaou 1996) there were different type of adatom events in this run047; hops (10 pieces), exchanges (2 pieces) and even one complex event including 3 atoms. The complex adatom event happened around timestep 52925100. The listing above shows also how many times the bondbreak algorithm has been in transition state before `offset` has increased, in other words how often $\eta \cong 1$ less than `waitTime` time steps. This is written as number of attempts in the listing. As can be seen there are hundreds or thousands attempts before `offset` is increased. Physically this tells that system is in a transition region, but falls back to a previous state just before moving permanently to a new state.

Similar analysis as above was done also for time steps that the `kCount.pl` script indicated as possible adatom events based on the LAMMPS dump file. With Ovito, it was seen that these time

steps were not associated with adatom events except when an event identified from the LAMMPS dump file was correlated with the time step when the event was identified from COLVARS trajectory file. In its current format `kCount.pl` cannot be used to identify adatom events from the LAMMPS dump file.

With long simulations, the LAMMPS dump file grows large and it is difficult to find interesting moments with Ovito without help of `kCount.pl` or similar tool that identifies interesting points in the simulation. When analyzing run047 it was validated with Ovito that the adatom was at the same place and with same atom Id after an adatom event identified from the COLVARS trajectory file and before the next adatom event. This is an evidence that there has been no adatom events that wouldn't have increased `offset`.

The conclusion is that `offset` value, extracted from bondbreak CV with equation 33, tells reliably the number of adatom events in the simulation. When `offset` is increased, there is always an adatom event and there are no adatom events where `offset` would not have been increased. The whole analysis of output of `kCount.pl` for run 047 is in an appendix B .

The second way to estimate, if CVHD has effect on the results of the simulation, is to analyze how close simulations follow Arrhenius' equation 4 with and without CVHD. This is done by drawing Arrhenius plot from simulations with and without CVHD in different temperatures.

The figure 9 follows equation 3, x-axis is inverse of the temperature and y-axis is logarithm of number of adatom events. The figure shows that the Arrhenius plot is in practice identical with and without CVHD, the blue line (without CVHD) and the orange line (with CVHD) are on top of each other with values $0.00167 \leq {}^{1}/_{T} \leq 0{,}0025$ (temperature range 400 K – 600 K). Both plots also follow closely a straight line (dotted line in the figure). It was not possible to get CV values without CVHD in a reasonable time for ${}^{1}/_{T} > 0.333$, which corresponds to temperature 300 K, and therefore blue line ends at that point. The run043, used to simulate adatom events in 300 K, terminated after 7 days and 12 hours with zero adatom events, CV was 0.587 when simulation ended. With lower temperatures and greater ${}^{1}/_{T}$ the simulations without CVHD would have taken even longer time before the first adatom event. This also explains why Arrhenius plot without CVHD differs from Arrhenius plot with CVHD at 300 K, there are statistically not enough adatom events for the simulation without CVHD for the result to be accurate.

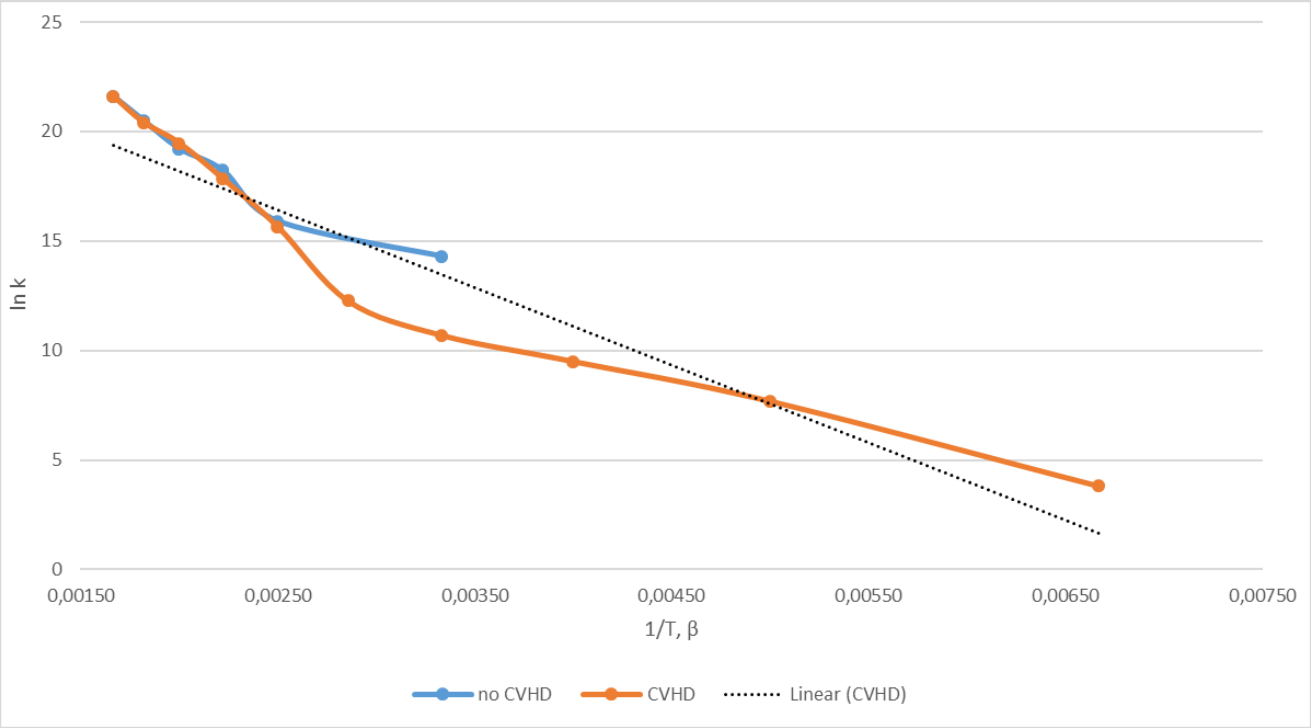

Figure 9: Arrhenius plot of simulations with and without CVHD

Data for figure 9 was collected from several simulations at different temperatures. These simulations were executed with a slab of size 6 x 6 x 5 and an adatom at the top of the slab. Between different simulations temperature and usage of CVHD was changed, but all other parameters were kept unchanged.

T	1/T = β	Run	Steps (*10^6)	CV	k (1/s)	ln k
600	0,00167	run045	300	1457	2428333333	21,61047
550	0,00182	run042	300	485	808333333	20,51049
500	0,00200	run046	300	133	221666667	19,21669
450	0,00222	run041	300	48,98	81633333,3	18,21775
400	0,00250	run044	289,499	4,662532	8052759,43	15,90153
300	0,00333	run043	182,89	0,586918	1604566,36	14,28836

Table 2: Simulation results without CVHD used for Arrhenius plot

T	1/T = β	Run	Steps (*10^6)	CV	k (1/s)	ln k
600	0,00167	run056	300,0	1437	2,39E+09	21,59663
550	0,00182	run057	301,5	441	7,31E+08	20,41028
500	0,00200	run054	342,1	194,9588	2,85E+08	19,46778
450	0,00222	run053	708,4	78,97538	55739842	17,83621
400	0,00250	run050	4 057,9	50,91503	6273635	15,65187
350	0,00286	run051	148 945,1	62,90333	211162,8	12,26038

300	0,00333	run049 (retry)	821 443,3	72,41116	44075,57	10,69366
250	0,00400	run059	4 948 860,0	132,9571	13433,11	9,505478
200	0,00500	run060	43 461 747,0	190,2123	2188,273	7,690868
150	0,00667	run062	2 752 213 300,0	252,2269	45,82255	3,824776

Table 3: Simulation results with CVHD used for Arrhenius plot

In tables table 2 and table 3 column *Run* refers to *Run Id* in the appendix A that lists executed simulation runs.

For the simulations without CVHD, the column *Steps* is the number of millions of time steps that the simulation run. With used units one time step is femtosecond, and so a million timestep is 1 ns and column *Steps* is directly simulation time in nanoseconds. The target was to run all simulations without CVHD for 300 ns but with low temperatures the simulations were terminated by timeout after the simulation time specified in *Steps* column. For the simulations with CVHD the column *Steps* is hypertime calculated by fix `timeboost` and extracted from the LAMMPS output file after the simulation. As explained in chapter 4.3.3, hypertime is expressed in this context in picoseconds. For easier comparison hypertime in picoseconds is converted to nanoseconds in table 3. Also, simulations with CVHD were run for 300 million timesteps but due to acceleration provided by CVHD the effective simulation time (hypertime) correspond to much greater number of simulation steps.

The value in the column CV is obtained from COLVARS trajectory file, chapter 4.3.4, for the last time step of the simulation. Event frequency (column *k*) is obtained by dividing CV value with 2, which gives number of adatom events per number of nanoseconds specified in *Steps* column, and then scaling to a full second by dividing with a time corresponding to *Steps* column.

Using least squares method, it is possible to obtain slope and y-intersect presented in the table 4 from the data in tables table 2 and table 3.

	Slope, s	Y-intercept, y
no CVHD	-4413.1159 ± 638.591597	28.25011 ± 1.483562
CVHD	-3539.3 ± 422.942	25.24227 ± 1.501239

Table 4: Slope and Y -intersect of Arrhenius Plot

From s and y it is easy to derive parameters Γ_0 and E_A of equation 4.

$$\Gamma_0 = e^y \tag{41}$$

$$E_A = -k_B s \tag{42}$$

	Γ_0	E_A
no CVHD	$e^{28.25 \pm 1.48}$ = 421.28 – 8187.84 GHz mean 1857.233 GHz	0.380293 ± 0.055 eV
CVHD	$e^{25.24 \pm 1.50}$ = 20.45 – 411.68 GHz mean 91.744 GHz	0.304993 ± 0.036 eV

Table 5: Pre-exponential factor and activation energy with and without CVHD

By substituting s and y from table 4 to equations 41 and 42 values in table 5 are obtained for pre-exponential factor, Γ_0, and for activation energy, E_A

Although the pre-exponential factor and activation energy with and without CVHD differ from each other, they have, however, similar magnitude. These simulated pre-exponential factors and activation energies don't correspond to values in literature because different types of adatom events are not differentiated from each other and because the size of the simulation system is not taken into account. In their paper, Bal and Neyts obtained values $\Gamma_0 = 54\ THz$ and $E_A = 0.53$ eV for adatom hop, and values $\Gamma_0 = 430\ THz$ and $E_A = 0.76$ eV for adatom exchange (Bal & Neyts 2015a).

The effect of the size of the simulated system can be seen from the figure 10, which shows Arrhenius plot when the adatom diffusion simulation was run with CVHD on a slab with size of 10 x 10 x 6.

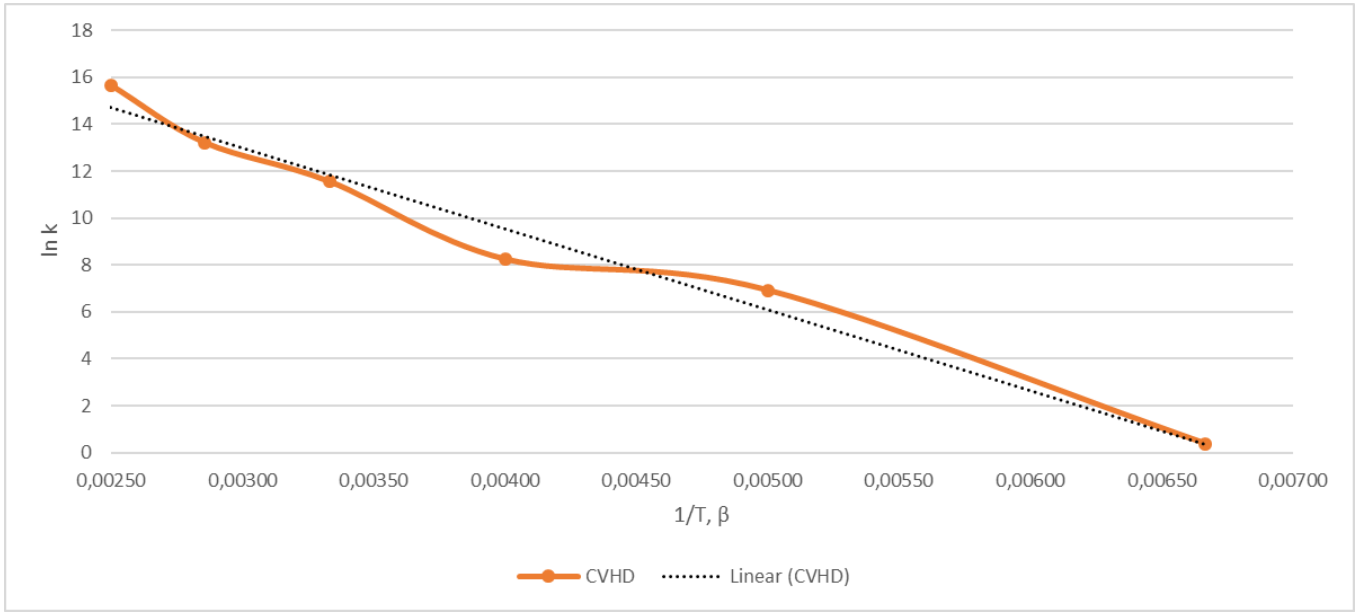

Figure 10: Arrhenius plot for CVHD simulation on a slab with size of 10 x 10 x6

The format of the curve is same in figures figure 9 and figure 10. With this bigger slab, the $\Gamma_0 = 13.78\ GHz$ and $E_A = 0.297 \pm 0.023$ eV. The activation energy corresponds to one in the table 5. With pre-exponential factor, there is a bigger difference between 6 x 6 x 5 and 10 x 10 x 6 slab, but this is due to exponential factor of this value, the difference in y-intersect is quite small. Y-intersect is 25.242 ± 1,50 for the smaller slab and 23.348 ±1.15 for the bigger slab. This shows that CVHD method gives the same result independent of the size of the simulated system.

A classical measure for the stability of the MD simulation is preservation and fluctuation of energy. This was measured by calculating standard deviation of temperature and total energy in percentage.

	Run		Std. Dev of Temperature [%]		Std. Dev of total energy [%]	
Temp	no CVHD	CVHD	no CVHD	CVHD	no CVHD	CVHD
150		run062		3.058 %		0.027 %
200		run060		3.062 %		0.036 %
250		run059		3.046 %		0.045 %
300	run043	run049 (retry)	3.043 %	3.048 %	0.051 %	0.053 %
350		run051		3.039 %		0.062 %
400	run044	run050	3.046 %	3.038 %	0.068 %	0.070 %
450	run041	run053	3.040 %	3.042 %	0.077 %	0.078 %
500	run046	run054	3.041 %	3.043 %	0.087 %	0.086 %
550	run042	run057	3.048 %	3.045 %	0.096 %	0.096 %
600	run045	run056	3.044 %	3.049 %	0.106 %	0.106 %

Table 6: Standard deviation of temperature and total energy with and without CVHD

As seen from table 6, standard deviations both with and without CVHD are small and values with and without CVHD are close to each other. This ensures that simulations with CVHD preserves physical quantities as well as simulations without CVHD.

To summarize it, the CVHD method doesn't change physical accuracy of the simulation. The bondbreak CV is a reliable measure for the number of adatom events. Simulations with CVHD produces similar Arrhenius plot as simulations without CVHD. The fluctuation of observed physical quantities are at same level in simulations with and without CVHD.

6.2 Boost achieved with CVHD

The acceleration achieved by CVHD method is measured with *boost* as described in chapter 3.3. The computational implementation for measuring boost is fix `timeboost` described in chapter

4.3.3. The fix `timeboost` writes hypertime to the LAMPPS log file. The *boost* can be calculated from there by dividing hypertime with classical simulation time (= number of time steps multiplied by length of the time step, dt). For example, run052 was run for 300 million timesteps, which correspond to 300 ns in this context and the run reported 148 945 070 ps as a value of hypertime. In this example boost is thus 148 945.070 ns / 300 ns = 496,5.

Run	Temperature (K)	Boost
run062	150	9174044.333
run060	200	144872.490
run059	250	16496.200
run049 (retry)	300	2738.144
run052	350	496.484
run050	400	13.526
run053	450	2.361
run054	500	1.140
run057	550	1.005
run056	600	1.000

Table 7: Boost of CVHD simulation for the slab of size 6 x 6 x 5

With CVHD quite substantial *boost* can be achieved at low temperatures, but at higher temperatures *boost* is so close to 1 that in practice there is no acceleration at all. Boost on table 7 were measured with CVHD simulation for the slab of size 6 x 6 x 5 and in this case CVHD provides real benefits only for temperatures lower than 400 K.

It can be seen from equation 31, that logarithm of the *boost* is proportional to the inverse of temperature. The figure 11 shows data from table 7 plotted this way.

With the bond boost method the logarithm of boost is directly proportional to inverse of temperature (Miron 2003). From the figure 11 it can be seen that also with CVHD the logarithm of boost is directly proportional to inverse of temperature when $^1/_T \geq 0.002857$ which corresponds to temperatures lower than 350 K. With higher temperatures, in this configuration, *boost* decreases faster than linear part of the plot.

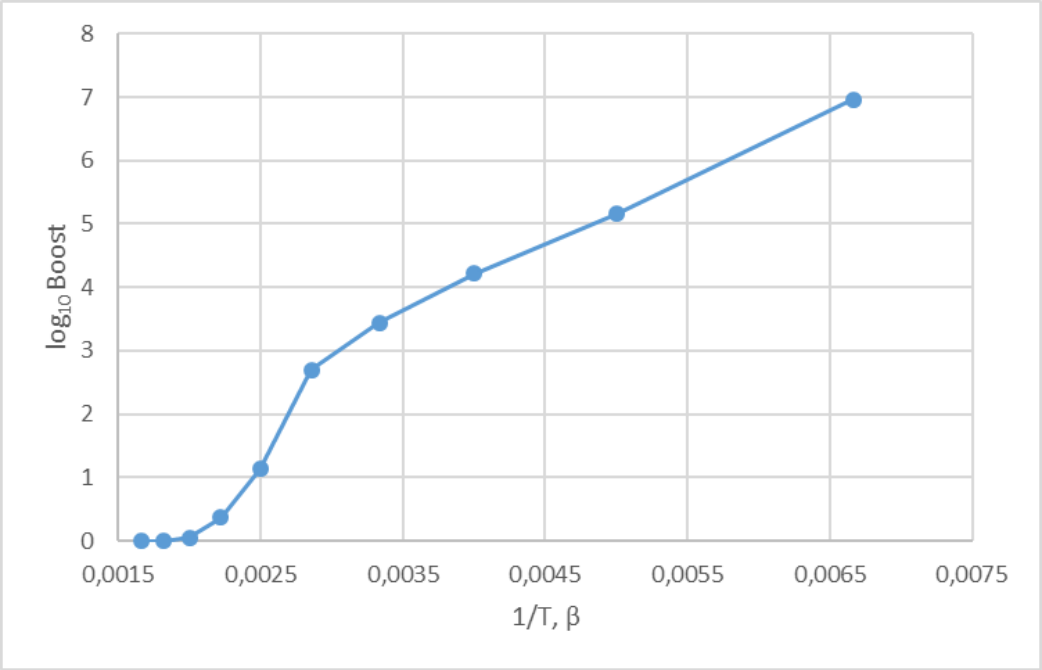

Figure 11: Logarithm of the boost versus inverse of temperature

Run	Temperature (K)	Boost	Γ (1/s)	Steps for 100 events	Boost 100 events
run062	150	9174044.333	0.310159	3.22416E+17	35 144 325 284
run060	200	144872.49	485.1819	2.06108E+14	1 422 687 484
run059	250	16496.2	40040.26	2.49749E+12	151 397 658
run049 (retry)	300	2738.144267	758971	1.31757E+11	48 119 211
run052	350	496.4835667	6207092	16110602976	32 449 418
run050	400	13.52619033	30018257	3331305960	246 285 604
run053	450	2.3614281	1.02E+08	977749840.2	414 050 227
run054	500	1.140363	2.73E+08	366706272.9	321 569 775
run057	550	1.005118167	6.08E+08	164379252	163 542 216
run056	600	1.000017233	1.19E+09	84227559.53	84 226 108

Table 8: Estimated number of simulation steps needed for 100 adatom events

When values from table 5 are inserted in equation 4, it is possible to calculate reaction rate Γ for different temperatures. In table 8 this is done using "no CVHD" values ($\Gamma_0 = 1857.23$ GHz and $E_A = 0{,}380293$ eV) from table 5. The inverse of Γ is an average time between successive adatom events and multiplying this average time with 100, it is possible to get estimate of simulated time needed for 100 adatom events. From this it is possible to calculate number of time steps of size of 1 fs needed to achieve this simulated time. This is column "Steps for 100 events" in table 8. And by

dividing number of 1 fs time steps with *boost* an estimate of number of simulation steps needed to achieve 100 adatom events with CVHD can be obtained, this is column “Boost 100 events” in table 8. By plotting number of needed simulation steps to achieve 100 adatom events in different temperatures with and without CVHD it is possible to see efficiency of CVHD method.

From figure 12 it can be seen that CVHD doesn’t provide any benefits for temperatures higher than 500 K, which can be seen also from table 7. But figure 12 shows, that there is also a lower limit for the usability of CVHD. With temperatures below 250 K the reaction rate, Γ, slows down faster than *boost* increases and thus number of simulation steps needed for 100 adatom events increases rapidly. With this configuration the usable range for CVHD method is from 250 K to 500 K.

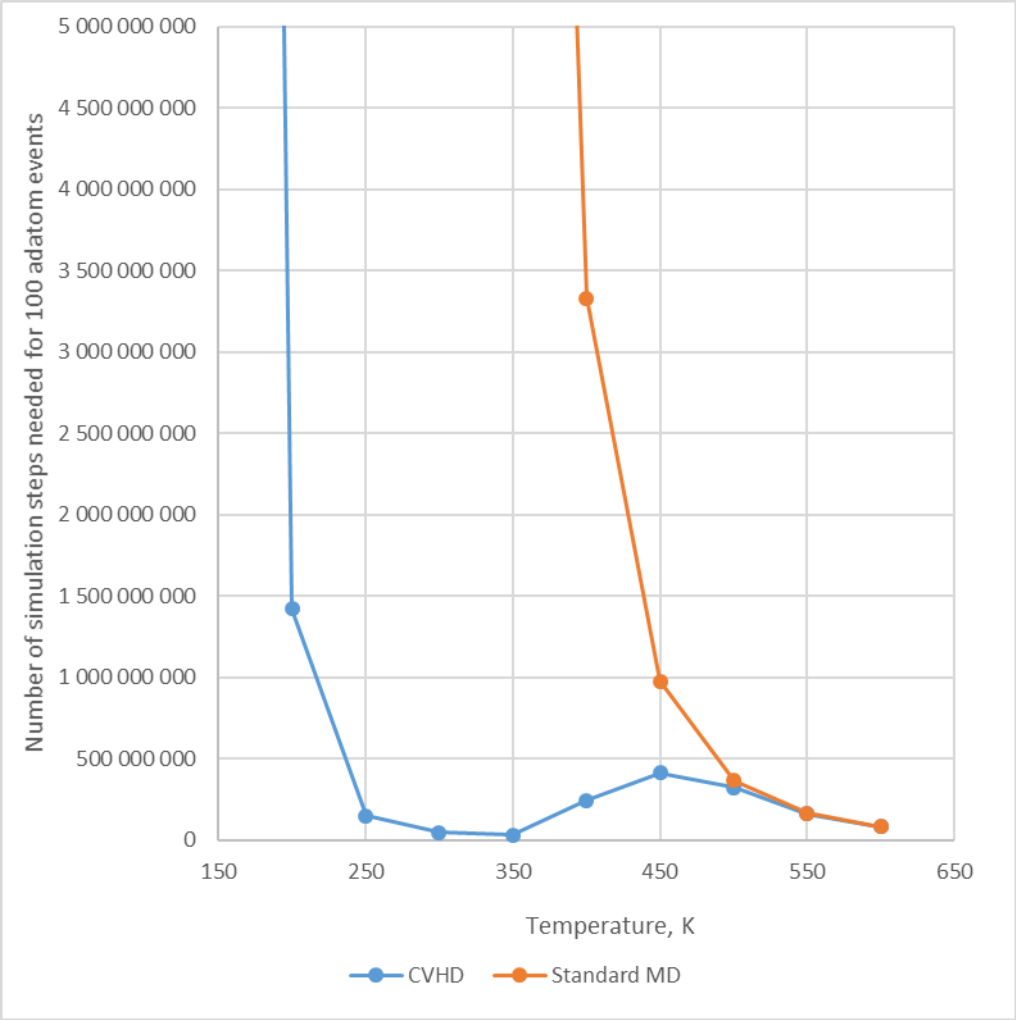

Figure 12: Comparison of number of needed simulation steps for 100 adatom events with and without CVHD

The effective use of CVHD method has thus a temperature range with upper and lower limit. This is the same phenomenon that Radu A. Miron and Kristen A. Fichthorn has described for the bond-

boost method (Miron 2003). It is evident from equation 31 that the *boost* declines with increasing temperature. Additionally, the average displacement of the atoms from equilibrium grows with increasing temperature and sampling of high-boost regions (around the local minimum) of the potential surface is reduced. On the other hand, at very low temperatures, transition times may increase faster than the boost factor and no transitions are observed over the simulation time scale. The method thus achieves peak efficiency in the midrange between the high-temperature domain of standard MD and very low temperatures, where kinetic Monte Carlo, kMC, is most efficient. (Miron 2003).

When the number of simulation steps needed for 100 adatom events with and without CVHD was compared the computer time use in simulation was not taken into account. CVHD is clearly slower than classical MD simulation, but this doesn't affect the magnitude of boost. According to LAMMPS log files, the simulation speed was over 5 000 time steps per second when simulations for lattice constant was run. When running CVHD simulation speed varied between 200 and 2 500 time steps per second. Even if CVHD simulation would be 25 times slower than classical MD, this is easily overcome by *boost* that can be from tens to millions.

All results so far in this chapter are based on simulations on the slab of size 6 x 6 x 5. To see the effect of the size of the slab simulations was also run on 10 x 10 x 6 slab. The first run with the bigger slab, run058, at 400 K show almost no *boost* with parameters used for the smaller slab. With bigger slab there are more atom pairs and thus χ_T, global distortion, goes near 1 easier even if none of bonds is broking but many of them are on mid-way r_i^{min} and r_i^{max}. It was possible to eliminate this effect by increasing parameter `power`. With value 20 of `power` it was possible to get similar result from the simulation of bigger slab than from the simulation of smaller slab.

Run	Temperature (K)	Boost
run068	150	156 920 384
run067	200	255 269
run066	250	38 952
run065	300	1 435
run064	350	234
run061	400	29.39

Table 9: CVHD boost for the slab of size 10 x 10 x 5

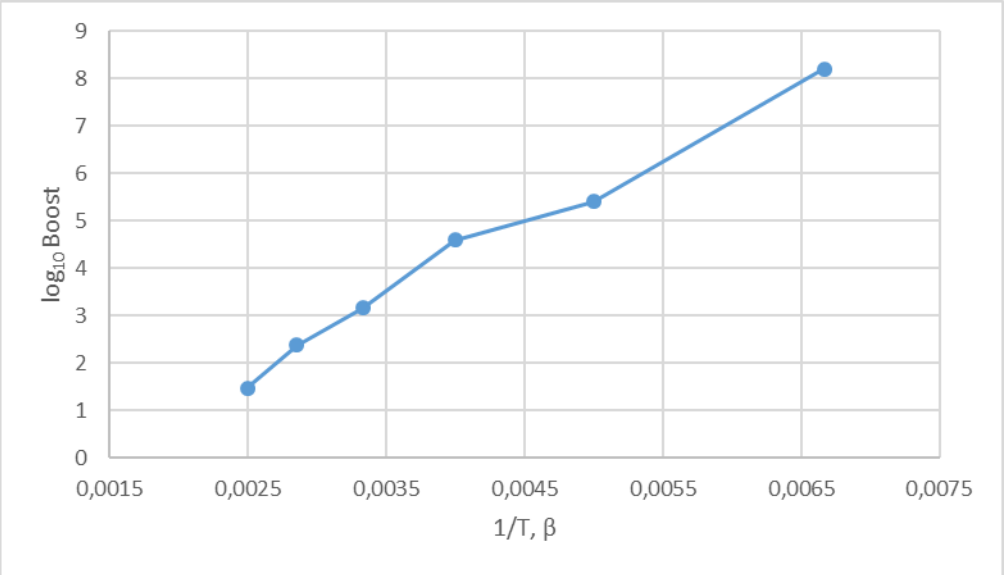

Figure 13: Boost of CVHD simulation of 10 x 10 x 6 slab

When the logarithm of the boost in table 9 is plotted against inverse of temperature in figure 13, it can be seen that the boost follows a straight line like with the smaller slab.

The conclusion is that boost achieved with CVHD is strongly dependent on temperature and there is both upper and lower limit for the temperature where CVHD is useful. Exact temperature limits and boost achieved depends on many parameters; physical system to be simulated, size of the system, computational resources used and so on.

	6 x 6 x 5 slab		10 x 10 x 6 slab	
Temperature (K)	Run	CV	Run	CV
150	run062	252.227	run068	22.9389
200	run060	190.212	run067	26.0945
250	run059	132.957	run066	14.9767
300	run049 (retry)	72.411	run065	15.000
350	run052	62.903	run064	13.000
400	run050	50.915	run061	18.9121

Table 10: Comparison of CV for different size of slabs at low temperatures

With low temperatures CV, which tells number of adatom events, increases, as shown by table 10. In this table each simulation was run for 300 million time steps for the smaller slab and 50 million time steps for the bigger slab.

Increasing CV with decreasing temperature doesn't fully align with earlier calculation in this chapter. According to table 8 number of time steps needed for a constant number of adatom events

should increase when temperature decreases which in other words means that CV should decrease with constant number of simulation step when temperature decreases. The estimated number of simulation steps needed for 100 adatom events corresponded better to the simulated CV when pre-exponential factor and activation energy were calculated from lower part of Arrhenius plot. In the context of this thesis, it was not possible to analyze in detail what causes this difference between estimated number of simulated steps and simulated CV.

7 Conclusions

With CVHD method it is possible to accelerate classical MD simulation by several order of magnitudes. However, there are both upper and lower limit for the temperature when CVHD is useful. In this thesis CVHD was used only to simulate adatom diffusion on copper (001) surface but theoretically CVHD should be suitable also for simulating other types of system. With other type of systems, the feasible temperature range will vary from that for the adatom diffusion.

For adatom diffusion simulations the bondbreak CV also provides a handy way to calculate number of adatom events.

The main restriction with CVHD method is earlier mentioned temperature range. Also, the size of atom groups used to calculate bondbreak CV afftects to the efficiency of CVHD, the computation time increases, when number of atoms in atom groups grows. The maximum size of the system used in simulations with this thesis was 2 401 atoms.

The current implementation of CVHD doesn't utilize parallelism: both CV in the `colvar::bondbreak` class and biasing force in the `colvarbias_cvhd` class are calculated serially in one thread. This can be seen by analyzing simulation speed (timestep/s) and time spent in modify tasks with different number of nodes used for simulation. This performance data is available from LAMMPS log file.

Run	Nodes	Timestep/s	Modify %	Modify ms/timestep
perf/n04	4	713.4703196	68.47 %	0.95967552
perf/n08	8	771.1527191	76.48 %	0.991762048
perf/n16	16	665.5485451	79.51 %	1.194653652

Table 11: Performance of CVHD with different number of nodes

As can be seen from table 11, the percentage of total time of simulation spent in modify tasks increases, when number of nodes increases, keeping milliseconds used for modifying at same level independent of number of nodes. The modify tasks include fixes, specifically `fix colvars` and `fix timeboost` in this context. This confirms what can be seen from source code, the current implementation of CVHD doesn't utilize parallel computing.

This is mentioned also in chapter 5.4 of Colvars manual "In simulations performed with message-passing programs (such as NAMD or LAMMPS), the calculation of energy and forces is distributed (i.e., parallelized) across multiple nodes, as well as over the processor cores of each node. When Colvars is enabled, certain atomic coordinates are collected on a single node, where the calculation of collective variables and of their biases is executed. This means that for simulations over large numbers of nodes, a Colvars calculation may produce a significant overhead, coming from the costs of transmitting atomic coordinates to one node and of processing them". (Bernardin, et al. 2020).

In this thesis CVHD was applied only with one setup, dynamic CVHD with well-tempered metadynamics. It would be interesting to see CVHD works both from the point of accuracy and efficiency with different setup/parameters, although the expectation is that different parametrization has only minor effect on CVHD.

Diffusion events can be considered as rare events. Rare events are also the idea behind transition state theory. In this sense it is understandable that simulations related to this thesis took considerable time to execute. Of the simulations that run 300 million timesteps with CVHD enabled the fastest was run054 (2 days 14 hours and 43 minutes corresponding to 1 328 time steps per second) at 500 K and the slowest was run052 (5 days 3 hours and 15 minutes corresponding to 676 time steps per second) at 350 K. The run length of over hundred million timesteps were needed in order to get statistically meaningful number of adatom events. This kind of simulations require thus quite a lot of time to complete.

8 Summary

The purpose of this thesis was to implement accelerated MD (Molecular Dynamics) method called CVHD (Collective variable-driven hyperdynamics) and analyze the suitability of this method for accelerating simulation of the surface diffusion on Cu(001) surface.

The original implementation of CVHD (Bal & Neyts 2015b) needed to be modified considerably to work with current (3 Mar 2020) version of LAMMPS. This modified version of CVHD is stored in version management system of University of Helsinki https://version.helsinki.fi/kurkmika/cvhd-for-lammps. With this modified version of LAMMPS it was possible to get similar results compared to the original version of CVHD (Bal & Neyts 2015a). It was validated that in the suitable temperature range CVHD provides significant acceleration compared to standard MD simulation while preserving physical accuracy.

CVHD uses collective variable, CV, to describe system's dynamics. The CV can be thought to be a generalized reaction coordinate. The acceleration provided by CVHD is based adding the bias potential energy, which in turn is based on the CV. CVHD has a modular design. Both CV and the biasing method based on CV can be chosen independently to be optimal for the system studied. In the context of this thesis only one CV, bondbreak CV, and one biasing method, dCVHD with well-tempered metadynamics, were used.

The CVHD method will be a valuable tool in the study of slow processes and rare events. (Bal & Neyts 2015a).

References

Bal, Kristof M. (2018). *E-Mail: Problems on Implementing CVHD.* .

Bal, Kristof M. & Neyts, Erik C. 2015a. Merging Metadynamics into Hyperdynamics: Accelerated Molecular Simulations Reaching Time Scales from Microseconds to Seconds. *Journal of Chemical Theory and Computation* 11: 4545-4554.

Bal, Kristof M. & Neyts, Erik C. 2015b. Source code of the CVHD implementation. https://pubs.acs.org/doi/suppl/10.1021/acs.jctc.5b00597/suppl_file/ct5b00597_si_001.zip Referenced Mar 21, 2020.

Barducci, Alessandro, Bussi, Giovanni & Parrinello, Michele. 2008. Well-Tempered Metadynamics: A Smoothly Converging and Tunable Free-Energy Method. *Physical Review Letters* 100: 020603.

Bennett, S. J. 1978. The Thermal Expansion of Copper between 300 and 700k. *Journal of Physics D: Applied Physics* 11: 777-780.

Bernardin, Alejandro, Chen, Haochuan, Comer, ComerJeffrey R., et al. 2020. COLLECTIVE VARIABLES MODULE
Reference manual for LAMMPS
Code version: 2020-02-25. https://lammps.sandia.gov/doc/PDF/colvars-refman-lammps.pdf Referenced May 25, 2020.

Britannica Academic. 2020. Arrhenius equation. https://academic-eb-com.libproxy.helsinki.fi/levels/collegiate/article/Arrhenius-equation/9619# Referenced 27 Mar 2020.

Callister, William D. Jr & Rethwisch, David G. 2014. *Materials Science and Engineering : An Introduction.* United States of America: Wiley.

CSC. FGCI - Finnish Grid and Cloud Infrastructure. https://research.csc.fi/fgci Referenced May 23, 2020.

Evangelakis, G. A. & Papanicolaou, N. I. (1996). *Adatom Self-Diffusion Processes on (001) Copper Surface by Molecular Dynamics.*

Fiorin, Giacomo, Klein, Michael L. & Hénin, Jérôme. 2013. Using Collective Variables to Drive Molecular Dynamics Simulations. *Molecular Physics* 111: 3345-3362.

Foiles, S. M., Baskes, M. I. & Daw, M. S. 1986. Embedded-Atom-Method Functions for the Fcc Metals Cu, Ag, Au, Ni, Pd, Pt, and their Alloys. *Phys.Rev.B: Condens.Matter; (United States)* 33: .

Frenkel, Daan & Smit, Berend. 2002. *Understanding Molecular Simulation : From Algorithms to Applications.* Computational Science (San Diego, Calif.). San Diego: Academic Press.

Gabriel, Edgar, Fagg, Graham E., Bosilca, George, et al. 2004. *Open MPI: Goals, Concept, and Design of a Next Generation MPI Implementation.*

Laio, Alessandro & Parrinello, Michele. 2002. Escaping Free-Energy Minima. *Proceedings of the National Academy of Sciences of the United States of America* 99: 12562-12566.

LeSar, Richard. 2013. *Introduction to Computational Materials Science : Fundamentals to Applications.* Cambridge ; New York: Cambridge University Press.

Miron, Radu A. 2003. Accelerated Molecular Dynamics with the Bond-Boost Method. *The Journal of Chemical Physics* 119: 6210-6216.

Mundy, J. N. 1981. Self-Diffusion in Chromium. *Phys.Rev.B: Condens.Matter; (United States)* 24: .

Plimpton, Steve. 1995. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics* 117: 1-19.

Plimpton, Steve. 2014. LAMMPS Features and Capabilities. http://lammps.sandia.gov/tutorials/italy14/italy_overview_Mar14.pdf Referenced Jun 25 2018.

Sandia National Laboratories. LAMMPS Molecular Dynamics Simulator. https://lammps.sandia.gov/index.html Referenced Mar 21, 2020.

Stukowski, Alexander. 2009. Visualization and Analysis of Atomistic Simulation Data with Ovitothe Open Visualization Tool. *Modelling and Simulation in Materials Science and Engineering* 18: 015012.

Voter, Arthur F. 1997. A Method for Accelerating the Molecular Dynamics Simulation of Infrequent Events. *The Journal of Chemical Physics* 106: 4665-4677.

Appendix A Table of simulation runs

The table below shows simulation runs for which this thesis is based on. For each run parameters used and results obtained are listed. The initial runs were used to get CVHD to work and later runs were used more systematically for measuring accuracy and efficiency of CVHD. The purpose of each run is commented after the parameter table.

Run Id	Cluster	Start date	Temp	Adatom	Dimensions	CV	CV	HD in effect	Steps	Wall time	timestep/s	Hypertime	Boost
run001	Alcyone	22.8.2018	300	bot	6 x 6 x 5	<1	on		50 000	0-02:10:55	6,3654	N/A	
run002	Alcyone	21.8.2018	300	top	6 x 6 x 5	<1	on		50 000	0-00:00:40	1250,0000	N/A	
run003	Alcyone	20.8.2018	600	top	6 x 6 x 5	81,5, non-broken bonds	on		50 000	0-00:00:36	1388,8889	N/A	
run004	Alcyone	22.8.2018	300	top	6 x 6 x 5	<1	on		150 000	0-09:00:00	4,6296	N/A	
run005	Kale	7.8.2018	600	top	4 x 4 x 3	242, no info about bonds	on		20 000 000	0-01:28:21	3772,8730	N/A	
run006	Kale	8.8.2018	600	top	4 x 4 x 3	242, no info about bonds	off		20 000 000	0-01:30:48	3671,0720	N/A	
run007	Alcyone	23.8.2018	300	top	6 x 6 x 5	<1	on		20 000 000	0-06:52:25	808,2441	N/A	
run008	Alcyone	24.8.2018	300	top	6 x 6 x 5	<1	off		20 000 000	0-04:36:11	1206,9278	N/A	
run009	Alcyone	24.8.2018	300	top	6 x 6 x 5	<1	on		20 000 000	0-12:04:04	460,3628	N/A	
run010	Kale	26.8.2018	300	top	6 x 6 x 5	<1	on		20 000 000	0-15:35:28	356,3284	N/A	
run011	Alcyone	28.8.2018	300	top	6 x 6 x 5	12, few indications of broken bond	md		20 000 000	0-12:46:08	435,0853	N/A	
run012	Alcyone	9.9.2018	300	top	6 x 6 x 5	< 1	on		20 000 000	0-06:52:09	808,7670	N/A	
run013	Alcyone	8.9.2018	300	top	6 x 6 x 5	4,94 no info about bonds	on		20 000 000	0-07:00:24	792,8957	N/A	
run014	Kale	15.12.2018	300	top	10 x 10 x 5	<1	on		20 000 000	0-13:43:27	404,8009	N/A	

run015	Alcyone	15.12.2018	300	top	10 x 10 x 5		on		20 000 000	>36:00:00	#VALUE!	N/A	
run016	Kale	16.12.2018	300	top	6 x 6 x 5	4,67 no info about bonds	on		20 000 000	0-06:18:50	879,8944	N/A	
run017	Kale	19.12.2018	300	top	6 x 6 x 5	<1	off		20 000 000	0-05:08:54	1079,0979	N/A	
run018	Kale	25.12.2018	600	top	6 x 6 x 5	87	off		20 000 000	0-02:59:54	1852,8812	N/A	
run019	Kale	27.12.2018	600	top	6 x 6 x 5	99	on		20 000 000	0-04:16:01	1301,9986	N/A	
run020	Kale	13.1.2019	600	top	6 x 6 x 5	99	on		20 000 000	0-04:21:20	1275,5102	0,423	0
run021	Kale	14.1.2019	600	top	6 x 6 x 5	7999	on		20 000 000	0-03:39:53	1515,9554	0,207	0
run022	Kale	14.1.2019	300	top	6 x 6 x 5	7999	on		20 000 000	0-04:11:52	1323,4516	0,446	0
run023	Kale	21.1.2019	300	top	6 x 6 x 5	7999	on		20 000 000	0-04:09:56	1333,6890	0,446	0
run024	Kale	22.1.2019	600	top	6 x 6 x 5	87	off		20 000 000	0-03:01:07	1840,4343	N/A	
run025	Kale	23.1.2019	600	top	6 x 6 x 5	9	on		20 000 000	0-01:47:27	3102,2181	59523,614	3
run026	Kale	24.1.2019	900	top	6 x 6 x 5	7337	off		20 000 000	0-02:02:11	2728,1408	N/A	
run027	Kale	25.1.2019	1200	top	6 x 6 x 5	6365	off		20 000 000	0-01:50:31	3016,1363	N/A	
run028	Kale	10.2.2019	600	top	6 x 6 x 5	87	off		20 000 000	0-03:02:43	1824,3182		
run029	Kale	24.2.2019	600	top	6 x 6 x 5	87	on		20 000 000	0-03:03:07	1820,3331	0,387	0
run030	Kale	6.3.2019	600	top	6 x 6 x 5		on		20 000 000				
run032	Kale	14.8.2019	600	top	6 x 6 x 5	87	on		20 000 000	0-03:08:48	1765,5367		
run033	Kale	16.1.2020	600	top	6 x 6 x 5	49,9	on		10 000 000	0-02:10:50	1273,8854	0,352	0
run034	Kale	17.1.2020	900	top	6 x 6 x 5	3661,5	on		10 000 000	0-01:54:14	1459,0020		
run035	Kale	20.1.2020	300	top	6 x 6 x 5	<1	on		20 000 000	#TIMEOUT	#VALUE!		
run036	Kale	20.1.2020	300	top	6 x 6 x 5	<1	off		20 000 000	#TIMEOUT	#VALUE!		
run037	Kale	20.1.2020	300	top	6 x 6 x 5	<1	on		20 000 000	0-08:52:55	625,4887	6,75E+13	3 374 907 950
run038	Kale	21.1.2020	300	top	6 x 6 x 5	<1	off		20 000 000	0-06:20:49	875,3118	20000,000	1
run039	Kale	21.1.2020	450	top	6 x 6 x 5	9,88E-01	on		20 000 000	0-03:42:19	1499,3628	7 043 191 600	352 160
run040	Kale	22.1.2020	450	top	6 x 6 x 5	48,98	on		300 000 000	2-18:06:48	1260,4618	817 861 960 000	2 726 207
run041	Kale	26.1.2020	450	top	6 x 6 x 5	48,98	off		300 000 000	5-05:27:12	664,2576	300 000	1

run042	Kale	4.2.2020	550	top	6 x 6 x 5	485	on		300 000 000	2-00:10:33	1729,7746	304 344	1
run043	Kale	7.2.2020	300	top	6 x 6 x 5	0,586918284	on		182 890 000	7-12:05:18	282,0992		
run044	Kale	11.2.2020	400	top	6 x 6 x 5	4,662531603	on		289 499 000	7-12:05:02	446,5496		
run045	Kale	22.2.2020	600	top	6 x 6 x 5	1457	on		300 000 000	1-11:30:53	2346,4447	4,230	0
run046	Kale	25.2.2020	500	top	6 x 6 x 5	133	on		300 000 000	1-13:55:00	2197,8022	278 814 550	929
run047	Kale	1.3.2020	500	top	6 x 6 x 5	26,995	on		100 000 000	0-14:51:50	1868,8096	234 430 790	2 344
run048	Kale	13.3.2020	550	top	6 x 6 x 5	155	on		100 000 000	0-13:42:55	2025,3165	100 992	1
Test/450K	Kale	15.3.2020	450	top	6 x 6 x 5	8,950356791	on	on	20 000 000	0-06:26:03	863,4460	42 097,604	2
run049	Kale	18.3.2020	300	top	6 x 6 x 5	20,7258303	on	on	93 519 000	1-14:08:04	681,2083	185 005 510	1 978
run050	Kale	19.3.2020	400	top	6 x 6 x 5	50,91503134	on	on	300 000 000	4-06:22:09	814,0472	4 057 857	14
run051	Kale	24.3.2020	350	top	6 x 6 x 5	62,90332746	on	on	300 000 000	3-11:16:44	1000,6538	148 945 070	496
run052	Ukko2	31.3.2020	350	top	6 x 6 x 5	62,90332746	on	on	300 000 000	5-03:15:36	676,0777	148 945 070	496
run049 (retry)	Kale	31.3.2020	300	top	6 x 6 x 5	72,41116248	on	on	300 000 000	4-19:32:27	721,2457	821 443 280	2 738
run053	Ukko2	5.4.2020	450	top	6 x 6 x 5	78,97537795	on	on	300 000 000	4-06:28:39	813,1866	708 428	2,36
run054	Kale	6.4.2020	500	top	6 x 6 x 5	194,9588178	on	on	300 000 000	2-14:43:34	1328,5270	342 109	1,14
run055	Kale	17.4.2020	550	top	6 x 6 x 5	Canceled	on	on	300 000 000	Canceled	#VALUE!	Canceled	
run056	Ukko2	10.4.2020	600	top	6 x 6 x 5	1437	on	on	300 000 000	3-17:55:54	926,6295	300005,170	1
run057	Ukko2	15.4.2020	550	top	6 x 6 x 5	441	on	on	300 000 000	3-23:47:29	869,9460	301535,450	1,01
run058	Kale	20.4.2020	400	top	10 x10 x 6	8,987657441	on	on	225 217 000	9-12:05:22	274,2796	227574,150	1,01
run059	Ukko2	24.4.2020	250	top	6 x 6 x 5	132,9571376	on	on	300 000 000	4-22:14:29	704,7730	4 948 860 000	16 496
run060	Ukko2	29.4.2020	200	top	6 x 6 x 5	190,2123161	on	on	300 000 000	4-14:01:09	757,4438	43 461 747 000	144 872
run061	Kale	1.5.2020	400	top	10 x10 x 6	18,91198669	on	on	50 000 000	2-00:47:38	284,6440	1 469 518	29
run062	Ukko2	6.5.2020	150	top	6 x 6 x 5	252,2268571	on	on	300 000 000	4-13:47:54	758,9672	2 752 213 300 000	9 174 044
run063	Kale	6.5.2020	400	top	10 x10 x 6	2,980685437	on	on	50 000 000	2-07:30:05	250,2440	50920,163	1,02
run064	Kale	9.5.2020	350	top	10 x10 x 6	13	on	on	50 000 000	2-03:33:01	269,4241	11 725 575	235
run065	Kale	11.5.2020	300	top	10 x10 x 6	15	on	on	50 000 000	4-02:19:20	141,2589	71 785 144	1 436

run066	Ukko2	11.5.2020	250	top	10 x10 x 6	14,97578081	on	on	50 000 000	2-11:47:04	232,3161	1 947 622 100	38 952
run067	Ukko2	14.5.2020	200	top	10 x10 x 6	26,09357228	on	on	50 000 000	2-09:06:55	243,1729	12 763 483 000	255 270
run068	Kale	15.5.2020	150	top	10 x10 x 6	22,93794495	on	on	50 000 000	3-01:03:42	190,0982	7 846 019 200 000	156 920 384
perf/n04	Ukko2	17.5.2020	350	top	6 x 6 x 5	7	on	on	25 000 000	0-09:44:00	713,4703	1 911 373	76
perf/n08	Ukko2	17.5.2020	350	top	6 x 6 x 5	8,404046676	on	on	25 000 000	0-09:00:19	771,1527	2 387 825	96
perf/n16	Ukko2	18.5.2020	350	top	6 x 6 x 5	8,599176844	on	on	25 000 000	0-10:26:03	665,5485	2 911 720	116

Run Id	Notes
run001	
run002	
run003	
run004	Partial run, terminated by timeout. Wall time is estimate
run005	
run006	Why CV is identical to run005. CVHD seem to have no effect
run007	
run008	Why CV is identical to run007. CVHD seem to have no effect
run009	Special run with lmp_serial. No parallelisation
run010	hillWeight 0,005 --> 5,000 no effect on CV, E_hd multiplied by 1000
run011	Test run with metadynamics instead of CVHD
run012	Test run using option unwrap no. Performance seem to be same than run_007
run013	Test run with lmp_mpi_kr. Run ended with segmentation fault, results still OK. Wall time not written in log.lammps
run014	Test to compare speed between kale and alcyone
run015	Test to compare speed between kale and alcyone. Cancelled due time limit
run016	

run017	Run to get comparison value of diffusion constant for run016
run018	Run018 and Run019 to get comparison between CVHD on/off at 600 K
run019	With CVHD the CV was 10 % larger than without it
run020	Test to see how hypertime calculation works. Hypertime boos smaller than expected and so msd[4] plot against hypertime didn't differ from plot against time
run021	Test with rmin = 0
run022	MSD plot is a horizontal line. It looks like no diffusion would happen. Out.colvars.traj is identical to run022 even if it should not be
run023	Added dump in order to get a video from the simulation. The video didn't show any adatom diffusion
run024	Test to see if diffusion constant can be calculated without CVHD. MSD plot was quite close to a line. Top adatom wouldn't move at all
run025	Test to see how CVHD works if only topatom is included in CVHD calculation and rmin = 0. Adatom didn't diffuse still but hypertime was huge
run026	Test to see if diffusion constant can be calculated without CVHD and if adatom would move
run027	Test to see if diffusion constant can be calculated without CVHD and if adatom would move
run028	Comparison to see if result is same than with run024 - the result was same. A new version (12Dec2018) of lmp_mpi was used
run029	Comparison to see if result is same than with run020 - it was not CVHD has no effect, but hypertime was advanced. A new version (12Dec2018) of lmp_mpi was used
run030	Comparison to see if result is same than with run020. A new version (12Dec2018) of lmp_mpi was used with older version of compbias_cvhd.cpp
run032	Performance comparison with run 029, when $WRKDIR is used instead of /tmp, no real difference on performance
run033	Added dump in order to get a video from the simulation and to see what happens when the value of CV changes
run034	Added dump in order to get a video from the simulation and to see what happens when the value of CV changes
run035	Comparison of CV value with and without CVHD. Simulation timed out in 6 hours and 14x10^6 steps
run036	Comparison of CV value with and without CVHD. Simulation timed out in 6 hours and 19x10^6 steps
run037	Rerun of run035 with longer run time and hypertime calculation enabled. CV of this run should be compared to CV of rerun of run 036
run038	Rerun of run036 in order to compare with run037
run039	A trial to compare CVHD and non-CVHD run in 450 K. This is not a good for comparison because CV < 1
run040	A trial to emulate results of Bal & Nyets. Extended the length of the run to $3x10^8$ (Bal & Nyets used value $2x10^8$)

run041	No difference in CV between runs 040 an 041 even if CVHD was used with run040. Strangely run041 was slower even if CVHD was not used with it
run042	A run to get more data for Arrhenius' graph
run043	#TIMEOUT: Terminated due to timelimit after 182 890 000 step, target was to run 300 000 000 steps
run044	#TIMEOUT: Terminated due to timelimit after 289 499 000 step, target was to run 300 000 000 steps
run045	Rerun of run 32 in order to CV value from a longer run for 600 K
run046	
run047	Run in 500 K with more trace to more exactly follow adatom trajectories
run048	Rerun of run 42 with different random seed (123456 --> 896520) and log in every 100 step. As expected the change of seed of the random number generator has not a big effect on results
Test/450K	First run after adding enable(f_cvb_apply_force); in init of CVHD bias. CVHD seems finally to work. CV is greater than Arrhenius' equation estimates for number of simulation timesteps. Arrhenius' estimation based on hypertime is CV = 5
run049	Run was interrupted by disk error, original setting was to run for 300 000 000 steps. **CVHD Works.** CV value is greater than in the run043 and is close to value 27,2 estimated by Arrhenius' equation
run050	CV is clearly bigger than in run 044
run051	
run052	
run049 (retry)	Rerun of run049 in order to get full simulation of 300 000 000 steps. After results were collecte they were accidentally overwritten by restarting this run again.
run053	
run054	
run055	This run was submitted already 9.4.2020, but due to Kale hardware problems the run was not started until 17.4.2020 and the run proceeded so slowly that it was canceled
run056	
run057	
run058	Test to see how the size of the slab affect to CV. Run was cancelled due to time limit. The target was to run simulation for 300 000 000 steps
run059	
run060	

run061	New test with big slab with p = 20 which favored large distortions better and boost was greater than with run 058
run062	
run063	Test to see what is boost when rMin = -1.0 and average bond length is used as rMin. Result, boost is almost same than with run058
run064	p = 20
run065	p = 20
run066	p = 20
run067	p = 20
run068	p = 20
perf/n04	Performance test to see what is timestep/s value when 4 nodes is used
perf/n08	Comparison value for run perf/n04 with 8 nodes
perf/n16	Comparison value for run perf/n04 with 16 nodes

Appendix B Output of kCount.pl

The analysis of the output of `kCount.pl` script for run047

```
Analysis of file atom.500K.dump and out500.colvars.traj at Tue Mar  3 16:52:18 2020
       Step  AdId               LatX               LatY               LatZ Message
       Step                 CV   Offset           Progress Message
          0                  0        0                  0 Bondbreak 0, offset increased after 0
stabilization Attempts - cleanlist initiated
          0   721                 3                  3           5.00001 AdAtom exchange 0: -1 --> 721

    6705300   721 AdAtom hop !!
    6707800                1.5        1               -0.5 Bondbreak 1, offset increased after 11588
stabilization Attempts - cleanlist initiated

    Ovito confirmed
    7399700   721            3.23804           3.83147           4.91472 AdAtom hop 1
    7402200                3.5        2              -0.5 Bondbreak 2, offset increased after 999
stabilization Attempts - cleanlist initiated

    7747500 Ovito adatom hop
    7749600                5.5        3              -0.5 Bondbreak 3, offset increased after 517
stabilization Attempts - cleanlist initiated

              Partial hop
    7965600   721            2.60228           4.78156           4.93152 AdAtom hop 2

              False positive
    9232700   721            2.57173           4.20722           4.99413 AdAtom hop 3
```

```
              False positive
   13479100   721            2.52057            4.75169            5.03165 AdAtom hop 4

              Partial hop
   13925900   721            2.46713             4.2295            5.01069 AdAtom hop 5

              Partial hop
   19044200   721            2.42203            4.75735             4.9564 AdAtom hop 6

   22088400  Ovito adatom hop
   22091000                7.5        4              -0.5 Bondbreak 4, offset increased after 19570
stabilization Attempts - cleanlist initiated

              Ovito Adatom Exchange 721 --> 360
   24385900   360             2.2247            5.72294            4.69786 AdAtom exchange 1: 721 --> 360
   24386000   721              1.676            5.40382            4.64501 AdAtom exchange 2: 360 --> 721
   24386100   155             2.6941           0.107311            4.71756 AdAtom exchange 3: 721 --> 155
   24386200   721            1.72321            5.37009            4.60686 AdAtom exchange 4: 155 --> 721
   24386300   155            2.72449           0.174282            4.65957 AdAtom exchange 5: 721 --> 155
   24386500   721            1.84739            5.34121            4.64578 AdAtom exchange 6: 155 --> 721
   24386700   360            2.22512            5.74689            4.58278 AdAtom exchange 7: 721 --> 360
   24386800   721             1.8347            5.25021            4.63836 AdAtom exchange 8: 360 --> 721
   24386900   344           0.900486            4.37181            4.63853 AdAtom exchange 9: 721 --> 344
   24387000   721             1.7637            5.27935            4.62807 AdAtom exchange 10: 344 --> 721
   24387100   355            1.31804            4.79634            4.62264 AdAtom exchange 11: 721 --> 355
   24387300   360            2.31618            5.67754            4.62751 AdAtom exchange 12: 355 --> 360
   24387400   155            2.78086           0.140702            4.63514 AdAtom exchange 13: 360 --> 155
   24387600   721             1.7656            5.29111            4.62352 AdAtom exchange 14: 155 --> 721
   24387800   355            1.42669            4.80304            4.66261 AdAtom exchange 15: 721 --> 355
   24387900   360            2.37186            5.56264            4.67703 AdAtom exchange 16: 355 --> 360
   24388000                9.5        5              -0.5 Bondbreak 5, offset increased after 3333
stabilization Attempts - cleanlist initiated

              Partial hop
```

```
   26338400   360             2.5266            5.77317            4.99102 AdAtom hop 7
   26338500   360            2.53634            5.65142             4.9374 AdAtom hop 8

              Partial hop
   27659900   360            2.50824            5.76985            4.97114 AdAtom hop 9
   27660000   360             2.3488            5.72157            5.00369 AdAtom hop 10

   27909500   Ovito adatom hop
   27909500   360            2.72245            5.80181            4.92102 AdAtom hop 11
   27912000               11.5        6                -0.5 Bondbreak 6, offset increased after 4582
stabilization Attempts - cleanlist initiated

              Partial hop
   29295900   360            2.96177            5.73346            4.99984 AdAtom hop 12
   29296000   360            2.99219            5.92086            4.90509 AdAtom hop 13

              Partial hop
   30155900   360            3.05125            5.72354            4.99004 AdAtom hop 14
   30156100   360            3.09707            5.93104             4.8884 AdAtom hop 15

              False positive
   31144100   360            3.03202            5.71508            4.98703 AdAtom hop 16
   31144200   360            2.92904            5.84779            4.91416 AdAtom hop 17

              False positive
   42859000   360            3.04886            5.72306            4.91062 AdAtom hop 18
   42859100   360              3.067            5.79358            4.91255 AdAtom hop 19

              False positive
   42941100   360            3.01938            5.70331            4.95364 AdAtom hop 20
   42941200   360            3.05552            5.86378            4.91598 AdAtom hop 21

              False positive
   46065900   360            2.97064            5.73081            4.87968 AdAtom hop 22
```

```
   46066000   360            2.99321             5.9147            4.90751 AdAtom hop 23

              False positive
   46881800   360            2.88805            5.74766            4.97824 AdAtom hop 24
   46882000   360            3.00325            5.86089            4.98731 AdAtom hop 25

   A complex event, visible also in Ovito
   52923000   507            3.64294           0.124562            4.65057 AdAtom exchange 17: 360 --> 507
   52923100   348            1.80587            4.40398             4.5965 AdAtom exchange 18: 507 --> 348
   52925100               13.5        7               -0.5 Bondbreak 7, offset increased after 36439
stabilization Attempts - cleanlist initiated

              Partial hop
   54258800   348            1.24301            4.39706            4.91066 AdAtom hop 26

              Partial hop
   57129100   348            1.77599            4.53515            4.99151 AdAtom hop 27

              Ovito Adatom Exchange 348 --> 355
   59043800   355            1.56937            5.28382            4.71711 AdAtom exchange 19: 348 --> 355
   59046600               15.5        8               -0.5 Bondbreak 8, offset increased after 8944
stabilization Attempts - cleanlist initiated

              Partial hop
   62042600   355            1.60237            5.75926            4.93523 AdAtom hop 28
   62042700   355             1.5561            5.66689            4.92021 AdAtom hop 29

              Partial hop
   65909600   355            1.55984            5.75041            4.92886 AdAtom hop 30
   65909700   355            1.62341            5.67187            4.93768 AdAtom hop 31

   66899800   Ovito AdAtom hop
   66902400               17.5        9               -0.5 Bondbreak 9, offset increased after 11242
stabilization Attempts - cleanlist initiated
```

```
             Partial hop
   67765200   355           0.743838           5.07257           4.94482 AdAtom hop 32

             Ovito AdAtom hop
   68524100   355            1.40745           5.43453           4.97359 AdAtom hop 33
   68526500              19.5       10              -0.5 Bondbreak 10, offset increased after 2330
stabilization Attempts - cleanlist initiated

             Ovito AdAtom hop or possible even double hop
   68698300   355            1.03073           5.95028           4.87233 AdAtom hop 34
   68700600              21.5       11              -0.5 Bondbreak 11, offset increased after 247
stabilization Attempts - cleanlist initiated

             Partial hop
   70821300   355           0.230689          0.593425           4.98846 AdAtom hop 35

             Partial hop
   70851300   355           0.800046           0.63666           4.95903 AdAtom hop 36

             Partial hop
   73472800   355           0.249506             0.551           4.88941 AdAtom hop 37

             Ovito AdAtom hop
   74252000   355           0.764472          0.699414            4.9193 AdAtom hop 38
   74254600              23.5       12              -0.5 Bondbreak 12, offset increased after 8001
stabilization Attempts - cleanlist initiated

             Ovito AdAtom hop
   78092800   355            1.28783           1.25162           4.97152 AdAtom hop 39
   78095200              25.5       13              -0.5 Bondbreak 13, offset increased after 5615
stabilization Attempts - cleanlist initiated

With visual analysis the adatom didn't move after that
```

```
Number of adatom hops is 39
Number of adatom exchanges is 19
Number of bond breaks is 13
Total number of adatom events is 58

Visual analysis with Ovito
10 adatom hops
2 adatom exchanges
1 complex event
13 in total
```