THE UNIVERSITY
of EDINBURGH

MIST: A Portable and Efficient Toolkit for Molecular Dynamics Integration Algorithm Development

Iain Bethune

Doctor of Philosophy
The University of Edinburgh
August 2020

Abstract

The main contribution of this thesis is MIST, the Molecular Integration Simulation Toolkit, a lightweight and efficient software library written in C++ which provides an abstract interface to common Molecular Dynamics codes, enabling rapid and portable development of new integration schemes for Molecular Dynamics. The initial release provides plug-in interfaces to NAMD-Lite, GROMACS, Amber and LAMMPS and includes several standard integration schemes, a constraint solver, temperature control using Langevin Dynamics, temperature and pressure control using Nosé-Hoover chains, and five advanced sampling schemes.

I describe the architecture, functionality and internal details of the library and the C and Fortran APIs which can be used to interface additional MD codes to MIST. As an example to future developers, each of the existing plug-ins and the integrators that are included with MIST are described. Brief instructions for compilation and use of the library are also given as a reference to users.

The library is designed to be expressive, portable and performant, and I show via a range of test systems that MIST introduces negligible overheads for serial, parallel, and GPU-accelerated cases, except for Amber where the native integrators run directly on the GPU itself, but only run on the CPU in MIST. The capabilities of MIST for production-quality simulations are demonstrated through the use of a simulated tempering simulation to study the free energy landscape of Alanine-12 in both vacuum and detailed solvent conditions.

I also present the evaluation and application of force-field and *ab initio* Molecular Dynamics to study the structural properties and behaviour of olivine melts. Three existing classical potentials for fayalite are tested and found to give lattice parameters and Radial Distribution Functions in good agreement with experimental data. For forsterite, lattice parameters at ambient pressure and

temperature are slightly over-predicted by simulation (similar to other reported results in the literature). Likewise, higher-than expected thermal expansion coefficients and heat capacities are obtained from both *ab initio* and classical methods. The structure of both the crystal and melt are found to be in good agreement with experimental data. Several methodological improvements which could improve the accuracy of melting point determination and the thermal expansion coefficients are discussed.

Lay Summary

Molecular Dynamics (MD) is a highly popular and versatile technique for modelling the behaviour of materials at the nano-scale. In classical MD, atoms are represented individually as point particles and interact with each other through simple spring forces - a crude model of a chemical bond between a pair of atoms - and the electrostatic force which exists between charged particles. For such a relatively simple model, MD is remarkable successful and has been used to model materials as diverse as biomolecules, semiconductors, crystalline minerals and exotic high-pressure phases of hydrogen.

The behaviour of the atoms over a period of time is simulated by taking small time steps (typically a millionth of a billionth of a second) where the atoms move before forces are recalculated. Using modern High Performance Computers, it is possible to simulate systems with up to millions of atoms for long enough to observe dynamical processes such as phase changes, protein folding and the deposition of molecules on surfaces. The output of an MD simulation is a series of snaphots like the frames of a movie, and can be analysed to compute measurable properties like densities, compressability, and molecular structures which can be compared with experimental data.

As MD has grown in popularity, a number of 'production quality' software programs have been developed - such as GROMACS, LAMMPS, Amber and CP2K - which represent many hundreds of person-years of effort adding new features, testing and improving reliability, and optimising performance. As a result these programs have become very complex and it is hard for researchers outside of the core development teams to understand and modify the source code to experiment with new features. This complexity barrier creates a catch-22 situation where potentially significant new developments are forced to be tested and demonstrated in simplified, lower-performance and sometimes private 'home-grown' MD programs. As a result, they do not receive wide adoption or attention

from the MD user community and so there is little demand for them to ever be implemented in the mainstream 'production' codes!

This thesis describes a software library called the Molecular Integration Simulation Toolkit (MIST) which lowers the complexity barrier for the development of new MD algorithms by providing an abstract interface which can be plugged-in to a range of the mainstream MD codes. This enables both the ease of development of 'home-grown' codes and the high performance, reliability and support for all of the surrounding software tools ecosystem of the 'production' codes in a single package.

I demonstrate that MIST achieves both these aims by implementing a range of recently developed methods for Molecular Dynamics, and providing performance benchmark data that shows a low performance overhead compared to using an MD code directly without MIST.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Parts of this work have been published in [24][25].

(*Iain Bethune, August 2020*)

Acknowledgements

Various people deserve mention for their part in supporting my long and winding road to thesis completion:

- Alison Kennedy, for first suggesting that I should consider a part-time PhD course as a way of building my research track record alongside my role at EPCC. By the time I made it to the end I was no longer working at EPCC - or even in a research organisation - but I'm still glad I took up the opportunity.
- Mark Bull, who advised me against doing the PhD, saying I would end up with all the hard parts of postgraduate study without being able to enjoy the good parts. I ignored his advice, but I have to admit he was right!
- Graeme, Ben and Chrystele for supervision throughout the whole process and constructive comments on the draft thesis. Thanks for bearing with me through three job changes, two relocations, and the resulting variation in the level of time and effort I was able to put in at different points over the years!
- Prof. Davide Marenduzzo and Dr. James Kermode for the detailed questions, comments and feedback they provided on the submitted thesis which (I think) greatly improved the final product.
- EPCC and the Hartree Centre, who supported me by covering my tuition fees, providing compute resources and (occasionally) time to work on my studies.
- Lastly and most importantly, my wife Julie, who has supported and encouraged me right through the 8 years it took to complete the PhD programme. I may have pushed her beyond the bounds of reasonable expectation by taking two weeks off at conferences while leaving her with a month-old baby and three other children in a house with no kitchen - but I hope you agree it was worth it in the end! Thank you for believing I could do it when I didn't quite believe it myself and keeping me focussed on finishing up over the final months.

This work used the ARCHER UK National Supercomputing Service (`http://www.archer.ac.uk`) and systems at the STFC Hartree Centre (`http://www.hartree.stfc.ac.uk`). This work used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1). Funding has been provided by the Engineering and Physical Sciences Research grants EP/K039512/1 and EP/P006175/1, an ARCHER eCSE grant, the University of Edinburgh via a Staff Scholarship, and by the Hartree Centre.

Contents

List of Figures

List of Tables

List of Code Listings

Chapter 1

Introduction

Molecular Dynamics has proved to be an extraordinarily successful method for studying dynamical processes as well as computing observables via sampling the conformational space of complex systems such as macromolecules (see [66] for a recent review). This success is largely due to advances in four directions; improving accuracy of force-fields, developing faster and more scalable force calculations, increasing computational power of high performance computing systems, and advanced sampling algorithms such as metadynamics [128], replica-exchange MD [199] and parallel tempering [92]. A number of highly-optimised MD packages such as NAMD [171], GROMACS [4] and CP2K [127] have been developed through many hundreds of person-years of effort which implement a range of different force calculation methods and time stepping schemes, and are able to run on a range of commodity (CPU clusters and GPUs) and special-purpose [188] hardware and represent. All of this functionality and performance comes at a cost in terms of code complexity, and even if an MD code is open source, in practice it is difficult for researchers to add significant new features without close collaboration with the main developers of the code.

The result is that the core algorithms used for MD time stepping evolve very slowly. Typical integration schemes are based on velocity Verlet or leapfrog integration, combined with one of several common thermo- or baro-stats [20, 102, 147, 164]. Recent innovation has centred on higher level methods for promoting space exploration [222] or modifying the potential energy surface to lower barriers between metastable states [91]. I argue that there is "room at the bottom"[1] for

[1] With apologies to Richard P. Feynman.

innovative methods which modify the core integration step to access larger time steps and/or improved sampling accuracy (e.g. [49, 63, 81, 132, 133, 154]) which have not yet been implemented in any widely used 'production' MD codes.

The *status quo* is a catch-22 for applied mathematicians: if new algorithms cannot be easily incorporated into widely used MD packages, then it is impossible to demonstrate their benefits on complex systems of practical interest. If such demonstrations are not achievable, there will be little interest from the MD user community, and there is no incentive for MD package developers to implement the new methods; in many cases algorithms are left 'on the shelf' for long periods.

This conundrum is what I set out to address by the development of MIST– the Molecular Integration Simulation Toolkit. MIST is a software library (available from `https://bitbucket.org/extasy-project/mist`) which can be easily interfaced to a variety of MD codes (currently GROMACS [4], Amber [48], NAMD-Lite [93], LAMMPS [174] and Forcite [30]) and which provides an abstract interface to the state of the system being modeled. This enables integration algorithms to be programmed without concern for the complexities of a typical MD code while only incurring low performance overhead.

Although the term "integrator" is used herein to describe timestepping procedures for MD, MIST is deliberately not restrictive regarding the types of equations of motion that can be simulated; many of the algorithms implemented introduce extended systems to modify dynamics and/or vary the temperature to promote improved sampling. While MIST currently supports a range of classical MD codes, the design of MIST is flexible enough that it could also be used for *ab initio* MD based on the Born-Oppenheimer approximation.

1.1 Structure of the Thesis

With the entire PhD programme taking eight years in total, it's not surprising that there were to be changes of direction along the way. Initially the aim of the project was the implementation and application of some *ab-initio* Molecular Dynamics methods under the guidance of Prof. Graeme Ackland to provide an atomistic level of explanation for experimental mineral physics results from group of Dr. Chrystele Sanloup. After a the first year and a half of study, the experimental co-supervisor moved away and I began collaborating with a group

of biomolecular scientists / biophysicists, applied mathematicians and computer scientists through the ExTASY (Extensible Toolkit for Advanced Sampling and analYsis) project. Among the many goals of that collaboration, one was making it easier to implement new algorithms at scale in existing classical MD codes. Thus the seed of the idea that eventually grew into MIST was born. With Prof. Ben Leimkuhler joining as a co-supervisor, the PhD project was refocussed away from applications of *ab-initio* MD towards development of a platform for algorithm developers to rapidly prototype new integration schemes.

The structure of this thesis mirrors the development of the project. Chapter 2 reviews a number of key concepts in Molecular Dynamics (both classical and *ab-initio*), techniques for achieving high performance in modern MD codes and a range of existing software packages that have similar goals to MIST. Chapter 3 describes the underpinning design principles of MIST, the architecture, library Application Programmer's Interface (API) and the implementation details, with aim of providing enough information to allow others to understand and extend the library in future. It also describes the range of functionality currently implemented in MIST as a reference to users. Chapter 4 covers the testing, validation and benchmarking done to ensure the correctness and performance of MIST. Based on these results, I review the extent to which MIST meets the design goals set out earlier. An extended description of the application of one of the algorithms implemented in MIST to an exemplar problem in biophysics (previously published in [24]) is included in Chapter 5. Chapter 6 covers the evaluation and application of classical and *ab-initio* MD to the computation of bulk properties and structure of olivine melts. While the key questions tackled in this phase of the project remain unresolved, the early results may prove instructive for others working in the area. As with any project, there are a number of areas in which MIST can be further developed, and even used to address some of the challenges raised in the previous chapter. These are laid out and the whole project summarised in Chapter 7.

Chapter 2

Background

This chapter serves to introduce the key concepts which underpin the rest of the thesis. I review the basic methodology of Molecular Dynamics and its implementation in both classical force-field and *ab-initio* varieties and discuss relevant properties of MD integrators. Techniques for achieving high performance using modern High Performance Computing hardware are described as they impact on some of the design choices taken with MIST. Lastly, I review existing software packages that attempt to address the same, or similar, challenges to MIST.

2.1 Molecular Dynamics

In addition to measuring the physical properties and structure of materials through experiments, the use of atomistic simulation as a complementary approach has grown dramatically in popularity over recent decades. Indeed, over 50% of compute time on ARCHER, the UK National Supercomputer (2013-2020), was spent on atomistic simulations. Around 30-40% was typically consumed by *ab initio* / electronic structure codes such as VASP, CP2K and CASTEP with the remainder consisting of a range of classical MD codes including GROMACS, LAMMPS, NAMD and DL_POLY. Similar statistics have been reported on other systems worldwide [113, 192]. Atomistic simulations have been successfully used to model materials as diverse as high-pressure hydrogen [208], solid-state catalysts [50], proteins [195] and even whole virii [74].

As the name suggests, an atomistic simulation models the individual atoms within a material. Given a method for computing the potential energy (and thus forces) corresponding to a particular configuration of atoms it can be used to calculate static and dynamical properties of a system in thermodynamic equilibrium. The state of the system is represented by a set of variables including particle positions and velocities which vary over time and mass, charge, species which are typically fixed for the duration of the simulation. For condensed phase systems, Periodic Boundary Conditions (PBC) are typically employed, where an individual unit cell with a relatively small number of atoms are modeled explicitly and stored and the cell is assumed to be replicated in each dimension to construct an infinite bulk system. The lattice parameters which describe the unit cell may be fixed or allowed to vary.

One of the most popular approaches used to compute energies is the classical atomistic force-field. Here interaction potentials are defined between individual atoms. In the simplest case these are a function only of the atomic species and interparticle separation. A simple example is the Lennard-Jones [112] potential,

$$V_{L-J} = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

where the repulsion which stops atoms overlapping (from Pauli exclusion) is modeled by the $1/r^{12}$ term and the longer-range attractive force (van der Waals / dispersion) by the $1/r^6$ term. In common with all potentials, Lennard-Jones has some free parameters (ϵ, σ) which must be fitted to data obtained from experiment or other more accurate form of calculation. Lennard-Jones is too simple to model the behaviour of complex materials, and many different potential models have been developed that combine various pair potentials (e.g. Buckingham's $V = A\exp(-Br) - \frac{C}{r^6}$ [43], Morse's $V = D_e(1 - e^{-a(r-r_e)})^2$ [155]) with terms depending on interatomic bond angles, and explicit contributions from electrostatic interaction of charged ions. The electrostatic interaction is particularly problematic for computer simulations because while the electrostatic potential drops off as $1/r$, its integral over an infinite periodic system diverges. As a result, various techniques have been developed for splitting the computation of the electrostatic potential into a short-range part which can be computed directly and long-range part which is computed in reciprocal space following a Fourier transform. The strategy was first developed by Ewald [68], and subsequently improvements (e.g. [67]) are now typically implemented in modern atomistic

simulation codes. In general, force-fields may contain additional multi-body terms, and especially in the field of biomolecular simulation a large number of complex force-fields have been developed (e.g. GROMOS [196], CHARMM [143], AMBER [55]) and are widely used.

Letting $\mathbf{r}_i$ represent the position of the ith atom and given a potential energy $U(\mathbf{r}_0, \mathbf{r}_1, ..., \mathbf{r}_n)$, it is possible to compute an 'equilibrium' structure by minimising energy with respect to the coordinates. This is usually referred to as a relaxation or geometry optimisation, and makes use of algorithms such as Conjugate Gradients or Broyden-Fletcher-Goldfarb-Shanno (BFGS) [42, 71, 83, 187] that find the lowest (local) energy configuration, provided that the initial coordinates are reasonably close to the minimum and the potential energy surface is smooth. As a minimum of the potential energy, the resulting structure is the $T = 0$ limit of the equilibrium solid, since no kinetic energy component is included. The equivalent problem of identifying minimum *free energy* structures at a finite temperature is much more challenging and requires advanced sampling approaches such as those as described in Section 2.1.3. It is also possible to include the cell dimensions (or lattice vectors) as variables and minimise the energy with respect to both the atomic positions and the cell size/shape at a fixed external pressure.

In addition to static structures, dynamical properties can be evaluated using Molecular Dynamics. More detail of the theoretical and implementation aspects of MD can be found in [8, 75], but the essential ideas are presented here. Given an initial state (particle positions and momenta) of the system at time t_0 we can integrate Newton's second law $\ddot{\mathbf{r}}_i = \mathbf{F}_i/m_i$ to obtain the state at time $t_0 + \delta t$. The subscript i indicates the particle index and so within a timestep each particle's position and velocity can be integrated independently, subject to the forces which have been obtained by differentiating the potential energy with respect to the particle positions, or in practice evaluated analytically from the force-field. The time evolution of the particles' positions and velocities are of course coupled through the force-field. If this procedure is repeated for many time steps, a trajectory (or history) of states is obtained. Different algorithms may be used for the integration, each with different properties and computational cost.

The simplest approach is Euler's method:

$$x_{n+1} = x_n + \dot{x}_n \delta t$$

However the error in the trajectory is proportional to the step size, and so a very small timestep is needed to achieve good enough accuracy for practical calculations. This would be prohibitively expensive computationally, and so alternatives such as Verlet integration [213] are used in practice. Higher-order methods following the Runge-Kutta scheme [125] have also been used for MD. The Runge-Kutta 4th order scheme extends Euler's method by evaluating forces at four points per timestep, and achieves a global error (that is the error accumulated over all the time steps) which is only $\mathcal{O}(\delta t^4)$. However, the Verlet scheme is most commonly preferred since it requires only a single force evaluation per time-step, but still achieves an error of $\mathcal{O}(\delta t^2)$. In addition it is also a *symplectic* method, meaning that while the trajectory may deviate from the 'exact' trajectory, there is a well-defined (perturbed) Hamiltonian (see [134], Chapter 5) which is conserved. Linear and angular momentum are also conserved and it has the desirable property that the integrator is *time-reversible*, since the equations of motion are themselves time-reversible, leading to long-term stability. Although many integrators are both, it is possible to be symplectic but not time-reversible, for example the semi-implicit Euler scheme:

$$v_{n+1} = v_n + \dot{v}_n \delta t$$

$$x_{n+1} = x_n + v_{n+1} \delta t$$

To calculate observables from a MD trajectory, one must ensure that the system is in a relevant thermodynamic ensemble. Common ensembles are outlined in Table 2.1. For simulations which produce results for comparison with experiment, the NPT ensemble is often used since it corresponds to the experimental conditions of fixed temperature and pressure. However, the canonical (NVT) and microcanonical (NVE) ensembles are much simpler to implement and are still widely used. Various schemes may be used to maintain the temperature including crude velocity rescaling and Berendsen's thermostat [20], which do not generate a true thermodynamic ensemble since kinetic energy fluctuations are unphysically damped; the methods of Andersen [9] and Nosé-Hoover [160, 161], where particles are able to exchange kinetic energy with an external heat bath of a fixed temperature; and Nosé-Hoover chains [147], where a chain of coupled thermostats is used to preserve *ergodicity* - the ability of the integrator (given enough time) to sample the entire phase space of the system.

Ensemble	Abbr.	Constants	Notes
microcanonical	NVE	No. Particles, Volume, Total Energy	
canonical	NVT	No. Particles, Volume, Temperature	Thermostat required.
isobaric-isothermal	NPT	No. Particles, Pressure, Temperature	Thermo- and baro-stats, cell volume may change.
grand canonical	µVT	Chemical Potential, Volume, Temperature	Particle exchange with external reservoir.

Table 2.1 *Common thermodynamic ensembles used in MD.*

All of the above integrators are deterministic - given a starting state and fixed time step, they will always product the same trajectory. It is also possible to construct schemes based on stochastic differential equations (see [135], Section 6.2) such as Langevin Dynamics. In addition to the force derived from the interatomic potential, Langevin Dynamics introduces a dissipative drag force and a random pertubative force acting on each particle independently. Conceptually, this is related to Brownian Dynamics, where macroscopic particles move within a liquid medium, experiencing friction and random 'kicks' from interactions with the molecules in the liquid. Clearly this makes no physical sense for a fully atomistic simulation without implicit solvent as there is no medium! However, if the intent is not to simulate the detailed Hamiltonian dynamics of the system but only to calculate observables Langevin Dynamics offers several key advantages. Technically, none but the simplest systems with a deterministic integrator are rigorously ergodic and ergodicity can be proven for Langevin Dynamics (see [135], Section 6.4.4). With careful construction of the perturbative force, the system will relax to equilibrium at a certain temperature and the NVT ensemble will be sampled [110], so Langevin Dynamics can be an effective alternative to the deterministic thermostats described earlier. Finally, the dissipative force damps out vibrations in the system, potentially allowing larger time steps to be accessed without suffering from resonances [132].

After the simulation has run for some time and reached an equilibrium state i.e. observable quantities fluctuate around a mean without long-term drift, then properties can be calculated. For some observable A in a system of N particles, the ensemble average is given formally by a probabilistically weighed integral over the entire phase space of the system:

$$\langle A \rangle_{ensemble} = \frac{1}{N! h^{3N}} \int \int A(\mathbf{p}^N, \mathbf{r}^N) \rho(\mathbf{p}^N, \mathbf{r}^N) d\mathbf{p}^N d\mathbf{r}^N$$

$\rho(\mathbf{p}^N, \mathbf{r}^N))$ is the probability density function defined over the phase space

(typically a Boltzmann weighting, where lower energy microstates are more probable, and hence have a larger weight). If the integration scheme used to propagate the system in time is ergodic i.e. all regions of the phase space are accessible, then the Ergodic Hypothesis holds, which states that:

$$\langle A \rangle_{time} = \langle A \rangle_{ensemble}$$

where

$$\langle A \rangle_{time} = \lim_{T \to \infty} \frac{1}{T} \int_0^T A(\mathbf{p}^N(t), \mathbf{r}^N(t)) dt$$

As a result, the expectation value or ensemble average of any observable (e.g. energies, structural parameters, lattice constants) can be calculated by averaging over a sufficiently long trajectory.

Classical MD has been applied successfully to a range of systems including examples of geophysical interest, relevant to the systems studied in Chapter 6. Pedone *et al* [168] constructed a force-field fitted to structural data of binary oxides and applied it successfully to a wide range of silicate crystals and glasses at ambient temperature and pressure. Similarly, Guillot and Sator [87] developed another force-field, fitted directly to structural and density data for silicate melts. This accurately reproduces the density, heat capacity and expansivity of a range of such melts at mantle temperatures and zero pressure. The same interatomic potentials were also used [88] at moderate pressures up to 20 GPa and found to be in good agreement with relevant experimental data. From the trajectory data the coordination numbers of various species were computed and it was shown that aluminium tends to increase in coordination with increasing pressure more rapidly than silicon.

Despite these successes, there are a number of drawbacks of force-fields. The first of these is that since the functional form of the potential is fixed and free parameters in the model are fitted to a particular set of data, all potentials have a limited *transferability*, that is if they are applied in conditions vastly different to those under which the fitting was performed, then unphysical results may be obtained. For example, Guillot and Sator [88] reported that their potential performed less well on iron-rich silicates (e.g. fayalite) or high silica content melts (e.g. Rhyolite). The second drawback is that some force-fields depend explicitly

on a defined bonding structure between the particles. For example the force-field described by Walker *et al* [215] contains an explicit 3-body harmonic potential constraining the angle between the predefined O-Si-O triplets to the tetrahedral angle of 109.47°. By construction this prohibits modeling any scenario where bonds may be broken or formed, for example if polymerisation of the silicate tetrahedra occurs in the melt via a bridging oxygen. Finally, since particles have a defined charge and electrons are not included in the model, scenarios involving charge transfer such as oxidation and bonding may not be simulated. To overcome these limitations it is necessary to use more accurate a range of more accurate and (usually) more computationally intensive models are needed where can capture electronic interactions such as bonding and polarisation and material properties are emergent, rather than being parameterised.

One such approach are *reactive* force-fields [138] such as ReaxFF [186] where the energy is still calculated as the sum of bonded and non-bonded interaction potentials. However, the bonding structure is computed on-the-fly as a function of the particle positions, enabling more complex chemistry such as forming of bonds during chemisorption of molecules on surfaces and catalysis. There also exist various *polarisable* force-fields [109, 201] which extend the Core-Shell approach used in Walker's potential by attaching dipoles to particles and solving not only for the energies and forces for a given configuration, but also the self-consistent set of dipole moments.

Machine Learning (ML) potentials [157] such as [181] and [84] have also recently gained popularity. These replace the specific functional form of the potential in classical MD with a *neural network* which given a set of particle positions can output energies and forces. The network is typically trained on forces obtained from *ab-initio* calculations, analagous to the parameterisation process of a classical potential. At runtime, calculation of forces is of comparable cost to a classical force-field. By removing the limition of functional form ML potentials offer greater transferability, and hybrid methods have also been developed [137] where an ML potential is complemented by *ab-initio* forces on-the-fly for configurations where the 'confidence' of the predicted forces is low!

2.1.1 ab initio MD

To accurately model the behaviour of materials down to the level of electrons requires the use of Quantum Mechanics, where the system is in general described

by some time-dependent wavefunction:

$$\Psi(\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_N, \mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n, t)$$

and the dynamics of the system is governed by the Time-Dependent Schrödinger Equation:

$$\hat{H}\Psi(\mathbf{r}, t) = i\hbar\frac{\partial}{\partial t}\Psi(\mathbf{r}, t)$$

where $\hat{H}$ is the Hamiltonian or total energy operator. Ψ is a function of all the atomic and electronic coordinates, and no general construction is known, so approximations must be made. Since nuclei are *much* more massive than electrons (typically 3-5 orders of magnitude), and consequently move more slowly, the Born-Oppenheimer approximation [36] states that we can consider the nuclei as fixed at some particular instant, and solve the Time-Independent Schrödinger Equation (TISE) for the electronic system within an external potential created by the nuclei. Thus we now only need to consider the simplified electronic wavefunction:

$$\Psi(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n)$$

then solve the eigenvalue problem:

$$\hat{H}\Psi(\{\mathbf{r}\}) = E\Psi(\{\mathbf{r}\})$$

and the nuclei can be treated as classical particles. Thus the basic steps in Born-Oppenheimer Molecular Dynamics (BO-MD) are as follows:

1. Solve the TISE for the electronic system to get a total energy as a function of the nuclear coordinates.

2. Update the positions of the nuclei by numerical integration, where the energy of the electronic system acts as a potential and the nuclear forces are given by:

$$\mathbf{F}_I = -\nabla_{\mathbf{R}_I} E(\{\mathbf{R}_I\})$$

3. Go to 1

Within this scheme all of the same integration algorithms, ensembles, thermostats, and analysis methods as for classical molecular dynamics can be applied. In this work, all *ab initio* calculations are carried out within the BO-MD framework. However, an alternative method based on the work of Car and Parrinello [47] also exists, although it has fallen out of recent use. In this case the equations of motion directly couple the electronic and ionic systems via fictitious dynamical variables which by construction propagate the electrons correctly into the ground state corresponding to the ionic coordinates at the next time-step. To maintain accuracy a much shorter timestep is typically used compared with BO-MD, although each step is cheaper since it does not require a full calculation of the ground state energy at each configuration.

Within the Born-Oppenheimer Approximation we must solve the TISE where the Hamiltonian consists of the electronic kinetic energy, the electrostatic potential energy of the electron-nuclei interaction, and the electron-electron interaction:

$$\hat{H} = \Big[-\frac{1}{2}\nabla^2 + \hat{V}_{ext}(\{\mathbf{R}_I\}, \{\mathbf{r}_i\}) + \hat{V}_{e-e}(\{\mathbf{r}_i\}) \Big]$$

Nevertheless, we still need to construct the many-body all-electron wavefunction. The method of Hartree and Fock [94] used a linear combination of products of single-particle wavefunctions known as a Slater Determinant. The first practical implementation of this *wavefunction theory* was made in the Gaussian code [99], and scales as $\mathcal{O}(n^4)$, limiting its use to relatively small systems. Thus further approximations are still desirable.

In 1964 Hohenberg and Kohn [101] showed that the external potential V_{ext} in the Hamiltonian is a unique functional of the electron density $n(\mathbf{r})$. This remarkable fact says that given the density (a 3-dimensional function), we can calculate all the properties of the system as if we knew the full ($3n$-dimensional) wavefunction. Additionally, they also proved that a variational principle applies for the total energy. If we define the Hamiltonian as a functional of the electronic density (split into the kinetic, electron-electron and external potential parts), then this functional is minimised by the electronic density which corresponds to the true ground-state wavefunction of the system. *viz.*

$$E\big[n(\mathbf{r})\big] = T\big[n(\mathbf{r})\big] + U\big[n(\mathbf{r})\big] + V\big[n(\mathbf{r})\big], \quad E\big[n'(\mathbf{r})\big] > E_0$$

In the following year Kohn and Sham [117] published a practical scheme for determining the ground state density. Instead of treating the many-body wavefunction directly, they considered a set of Kohn-Sham orbitals (or wavefunctions) corresponding to fictitious, non-interacting electrons moving in an effective potential given by:

$$V_{KS}(\mathbf{r}) = V_{ext}(\mathbf{r}) + V_H(n(\mathbf{r})) + V_{XC}(n(\mathbf{r}))$$

Where the External potential is as before, the electron-electron (Hartree) potential is integrated over the electron density:

$$V_H(\mathbf{r}) = \int \frac{n(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}'$$

and V_{XC}, the exchange-correlation potential is a small, unknown functional which contains all the electronic interactions which are ignored in the non-interacting Kohn-Sham scheme. Thus instead of having to solve a Schrödinger equation for a many-body system, the problem is simplified to solving several single-particle equations. In metals particularly, the number of single-particle equations needed is prohibitively large, so a single unit cell with k-point sampling (see e.g. [14]) is typically used in practice. Since the KS potential depends on the electron density, which itself depends on the potential, a procedure called Self-Consistent Field (SCF) iteration is performed, where given an initial guess at the density, we compute the resulting KS potential. The single-particle KS equations are solved:

$$\left[-\frac{1}{2}\nabla^2 + V_{KS}(\mathbf{r}) \right] \psi_i(\mathbf{r}) = \epsilon_i \psi_i(\mathbf{r})$$

to obtain the KS orbitals $\psi_i(\mathbf{r})$ and the resulting electronic density given by:

$$n(\mathbf{r}) = \sum_i |\psi_i(\mathbf{r})|^2$$

Due to the variational principle of Hohenberg and Kohn, the SCF procedure will

eventually converge such that the density is unchanged (within some tolerance, or machine precision), at which point we can terminate and compute energies, forces and properties of the system. The SCF procedure and the Kohn-Sham equations form the cornerstones of *Density Functional Theory* (DFT) and the method has seen rapid growth in usage across fields as diverse as solid-state and condensed matter physics, biosciences, materials chemistry, and chemical engineering.

2.1.2 Implementation: Basis Sets and Psuedopotentials

Within the framework of KS-DFT, a large number of computer programs (nearly 100 in mainstream use according to [2]) have been developed. One of the key features which differentiates these programs is the implementation choice of Basis Set used for representing the wavefunctions (or KS-orbitals). The orbitals are formally defined over all space, but in practice are localised around atoms, and so the question arises naturally of how they can be stored and operated on in a computer. Ideally, this representation should be compact in memory, efficient for computing derivatives and other quantities like the KS Hamiltonian, and accurate. Note that in addition to being wasteful of memory simply discretising the orbitals on a global grid does not give accurate enough derivatives for the kinetic energy term without an extremely fine grid. Typically the wavefunctions are expanded in a series of some analytic functions, and only the coefficients are stored i.e. just the set of c_i from:

$$\psi(\mathbf{r}) = \sum_i c_i \phi(\mathbf{r})$$

One of the most popular choices of basis set is to use plane-waves which are particularly convenient for the simulation of periodic solids and are amenable to parallisisation (see [167] for full discussion). In a plane-wave basis the wavefunctions are expanded as a Fourier Series and the coefficients of the series are stored in a regular grid, up to a maximum wave-vector called the plane-wave cutoff. The Kinetic Energy, Hartree Energy and external Potential Energy are typically computed in Fourier space, and a Fourier transform is used prior to computing the Exchange-Correlation Energy, and to evaluate the real-space density if required. During the SCF procedure the KS equations may be solved by matrix diagonalisation, at a cost of $\mathcal{O}(n^3)$, where n is the number of electrons, and so it is still desirable to reduce n further.

Other notable alternative basis sets are atom-centred Gaussians. Typically, fewer basis functions are required than for plane-waves for a given accuracy, and derivatives, products etc. can be computed analytically. However, because the basis set depends on the ionic coordinates, $\frac{\partial \psi}{\partial \mathbf{r}_i} \neq 0$, the Hellman-Feynman theorem [70] which is normally used to compute ionic forces does not hold:

$$\frac{\partial}{\partial \mathbf{r}_i} < \psi|H|\psi > \neq < \psi|\frac{\partial H}{\partial \mathbf{r}_i}|\psi >$$

The additional terms in the derivative are termed Pulay forces [177] and must be accounted for to generate correct dynamics with a Gaussian basis set approach.

It is also possible to represent the orbitals with a direct numerical approximation, for example on a regular grid with a small enough grid spacing to capture rapid variations accurately e.g. [156]. These approaches are usually combined with the Projector Augmented Wave (PAW) [33] or Linearized Augmented Planewave (LAPW) [193] method which allows the orbitals to be partitioned into an 'atomic' part localised to a sphere around each atom, where a fine grid (or other accurate representation) is used for the orbital, and everywhere else a 'smooth' part, represented by a small number of plane waves, for example. There are many additional basis sets which are less widely used including wavelets [78], Linear Muffin-Tin Orbitals (LMTO) [166], and various numerical approaches [34, 103].

The final approximation described here is the 'frozen core', that is to say that since only the valence (outer) electrons contribute to the chemistry of an atom, we can consider the inner electrons to be fixed and avoid modelling them directly. Instead we replace the coulombic potential of the nucleus with a *pseudopotential*, which represents the screened interaction of the valence electrons with the nucleus and all the core electrons together. Various approaches to constructing pseudopotentials are used including Ultra-soft [209], and others which introduce relativistic effects [15] and non-linear corrections for overlap of core and valence electron orbitals [142]. As well as reducing the total number of electrons (and hence the number of KS equations to solve), this has an additional benefit of removing the sharp peaks in the wavefunctions associated with the core electrons, and so less basis functions are needed to accurately represent the KS orbitals. The use of pseudopotentials as well as a plane-wave basis set is implemented in codes such as CASTEP [52], VASP [122], Quantum Espresso [79], CPMD [1], and many others.

2.1.3 Integrators

Whatever appropriate method is chosen to calculate forces, for practical calculations the choice of integration schemes. As discussed, deterministic integrators should be *symplectic* and *time reversible* - leading to long-term stability of the calculation - and ideally *ergodic* (or as close as possible for in practice), in order to allow accurate observables to the computed. While achieving a full sampling of a $6N$-dimension (N the number of particles) phase space is impractical given finite compute time, even achieving a good approximation to it is problematic. Many systems of interest - even simple molecules such as alanine dipeptide (see e.g. [96]) - are characterised by several local potential energy minima, separated by large energy barriers. Given that the probability of a system in the canonical ensemble occupying a microstate with energy E is proportional to e^{-E/k_BT}, at a finite temperature T the expected kinetic energy fluctuations are of order k_BT and so the system is highly unlikely to spontaneously transition from one minima to another. Intuitively, where many physical or chemical processes occur on timescales of microseconds or longer and molecular dynamics operates with timesteps of femtoseconds (to reduce discretisation error), extremely long calculations [189] are expected to be needed to observe even one barrier crossing - never mind enough to obtain a well-converged sampling for calculation observables.

To overcome this limitation, a wide range of so-called *enhanced sampling* [21] methods have been developed that enable simulations to cross potential energy barriers more frequently and unlock the energy landscape of complex systems to practical calculations. One approach is to modify the potential energy surface to reduce the effective height of the barrier. Umbrella Sampling [126] constrains the system via the introduction of a localising potential (the umbrella) to keep the system from dropping into the nearest minima - after sampling in several umbrellas along the expected reaction pathway, the free energy profile can be reconstructed by methods like WHAM [123]. Metadynamics [128] dynamically updates the potential energy surface by 'depositing' additional contributions to the potential energy that incrementally 'fill in' minima as the simulation progresses until the system spontaneously leaves the minima by crossing the lowest available barrier. Accelerated MD [91] directly modifies terms in the force-field (e.g. dihedral angle terms in biomolecular force-fields) to reduce the height of the barriers.

Another set of approaches are based on tempering - injecting kinetic energy into the system by raising the temperature to increase the rate of barrier crossing. Common schemes include Replica-Exchange MD [199] and Parallel Tempering [92] where a set of multiple simulations are run at different temperatures and configurations are periodically swapped. Simulated Tempering [145] extends this by allowing the temperature of a single simulation to evolve as a dynamical variable. Similarly to Umbrella sampling, a reweighting scheme is required to compute observables at a fixed temperature from the set of samples taken from a range of different temperatures. Recent developments in this area include Continuous Tempering [81] and Infinite Switch Simulated Tempering [146] both of which are discussed now implemented in MIST and described in Section 3.3.

A more recent innovation has been the development of adaptive sampling methods, whereby a number of short simulations are run from a given starting point e.g. a local minima. After a while, the set of states visited is analysed by a method such as Markov State Models [173], Principal Component Analysis [191] or Diffusion Maps [222], and based on the knowledge obtained of the shape of the free energy landscape a new set of starting points for additional simulations is selected to promote undersampled regions. These methods are becoming increasingly popular since running a set of many loosely coupled smaller simulations makes more efficient use of a parallel computer than running a single larger simulation [104].

There is also innovation at the level of timestepping algorithms - particularly in the construction of sympletic schemes for sampling from the canonical and isobaric-isothermal ensembles which allow for longer timesteps without adversely affecting configurational averages e.g. [49, 63, 132, 133, 154]. While these algorithms are all several years old, they have not yet been implemented in mainstream MD codes. I contend that for the purposes of rapidly developing and testing new algorithms, the authors typically implement these in home-grown codes which lack features and performance compared to 'production' codes - source code may not even be made available. The algorithms are then demonstrated on 'toy' problems such as small molecules, simple crystalline solids, perhaps with unrealistic (but easy to implement and fast to compute) force-fields. As a result of these factors, algorithms languish 'on the shelf' without being applied to real-world, complex systems that would demonstrate their practical benefits and lead to implementation in mainstream codes.

2.1.4 Parallelisation

Alongside the growth in usage of MD, all popular MD codes have implemented some form of parallelisation to harness modern computer hardware to perform multiple operations simultaneously, enabling simulations to be completed more rapidly, or larger simulations to be performed with fixed computational resources. The two key computational tasks which take time in an MD calculation (neglecting issues such as I/O, although this is also an active area of research [46, 162, 204]) are the calculation of the forces $\mathbf{F}_i$ on each particle, and the time integration of the particle positions and velocities. For all but the simplest force-fields the force calculation dominates the runtime, particularly for *ab-initio* MD where the cost of the force calculation for n particles could be $\mathcal{O}(n^3)$ or greater. Even for classical calculations using short-range cutoffs or summation methods like Particle-Mesh-Ewald [67] where the cost is $\mathcal{O}(nlogn)$ calculating the forces is usually the performance bottleneck.

Given the forces, time integration is trivially parallelisable over space since there is no dependency between updates to each particle i - so each processor can simply compute the position and velocity updates of a sub-set of particles. However, calculation of the force acting on a particle typically introduces some data dependency. For example, a simple pair-potential acting within a cutoff-range r_{cut} calculating the force on particle i:

$$F_i = -\frac{1}{m_i} \sum_{j, r_{ij} \leq r_{cut}} \frac{\partial U_{ij}}{\partial r_{ij}}$$

requires knowledge of the positions of all the particles j within the cut-off radius. In DFT, the per-particle forces are a functional of the electron density, which depends on the positions of all the particles in the system!

Typically, a choice is made between having the particle data (positions, velocities etc.) either *replicated* or *decomposed*. Replicated data has the advantage that every processor has access to all the data required to calculate forces but has the downside that expensive all-to-all communication is required after every time-step. It also creates a memory bottleneck as the number of particles which can be stored is limited to the memory of a single processor. Replicated data is most widely used in *ab initio* codes where the number of particles is typically less than 1000 and more complex parallelisation schemes are used to decompose the electron

density, KS orbitals and operator matrices. Most classical MD codes make use of a data decomposition scheme - either dividing the list of n particles evenly between processors, or more commonly employing a domain decomposition scheme where each processor 'owns' a region of the overall simulation space and stores and updates the particles that are located in that region. As the simulation progresses, particles may well move from one region to another, and so the decomposition may be adjusted periodically via a load-balancing algorithm with the aim of achieving roughly the same number of particles per processor (a common proxy for the amount of computational work to be done). Every MD code has subtly different decomposition and load-balancing schemes, for example LAMMPS supports both regular 3D 'brick' decompositions and irregular decomposition based on Recursive Coordinate Bisection - and can even load balance systems with different force-fields per particle [29]! NAMD takes a different approach and divides space into many small cubic patches ($n_{patches} >> n_{processors}$) and dynamically assigns patches to processors based on measured load imbalance as the simulation progresses [170].

OpenMP

Modern HPC systems are typically composed of many 1000s of compute nodes, connected via a high-speed interconnect. Within a node, memory is shared between a handful of multi-core processors - 12-64 cores per node is typical.

OpenMP [163] is a standardised language extension for C and Fortran to enable shared-memory programming that is supported by all major compilers (GCC, Intel, Cray etc.). OpenMP *directives* are added to serial code to introduce regions of thread parallelism, where multiple independent threads operate simultaneously. Individual pieces of data (variables, arrays etc.) can be marked as shared, in which case all threads may read or write the data, or private, in which case each thread has its own copy of the data. OpenMP is most commonly used to implement loop-level parallelism, where there is no dependency between different iterations of a loop and so they may be executed concurrently by multiple threads. For example, a loop performing a simple element-by-element averaging of two arrays could be parallelised with OpenMP as:

Listing 2.1: Example of an OpenMP-parallelised loop.

```
#pragma omp parallel for shared(a,b,c)
for (int i = 0; i < n; i++){
  a[i] = 0.5 * (b[i] + c[i]);
}
```

As a shared-memory programming model, no data decomposition is required, and so OpenMP has the advantage that it can be added incrementally to accelerate performance-critical regions of code. On entry to an OpenMP region, threads are spawned, carry out work in parallel to obtain a speedup and outside of these regions only a single thread continues execution. The main limitation of OpenMP is that it can only operate within a single compute node, so can only speed up performance by (at best) the number of available processing cores on a node. Further scaling up requires a programming model that can handle distributed memory.

Recent versions of OpenMP also allow for thread-parallelism on co-processor devices such as GPUs (see Section 2.1.4).

MPI

To enable a program to scale beyond a single node of an HPC system, a method of exchanging data across the network is required. MPI [150] is a standardised Application Programmer's Interface (API) that is implemented by a number of open-source and proprietary libraries e.g. OpenMPI, MPICH, Intel MPI, Cray MPT. In the distributed-memory programming model, each process (running on a single processor core or set of cores) has its own discrete memory space and may only exchange data with other processes by explicit calls to the MPI library. MPI provides functions for point-to-point communication between pairs of processes and collective communications between many or all processes such as global reductions or data broadcasts.

In Molecular Dynamics, MPI is most commonly used as part of a domain decomposition approach. For short-ranged forces it is necessary to access the positions of nearby particles, some of which may reside in another processor's memory. Nearest-neighbour point-to-point communication is typically used every timestep to update the positions of particles in a 'halo' region surrounding the core domain owned by each processor. The size of the halo is set large enough so that the positions of all particles which are needed for the force calculation (typically

those within a cut-off distance) are communicated. Global communication is minimised and used only for operations such as computing the total energy or other global quantities like pressure.

MPI may be combined with OpenMP in what is referred to as *mixed-mode* parallelisation, where multiple OpenMP threads are used within a node and MPI is used to communicate between nodes. This can provide better overall scalability in some cases, at a cost of increased code complexity [23].

GPUs

Since the late 2000s, Graphical Processing Units (GPUs) have become a viable option to accelerate the performance of HPC applications. A comprehensive overview of the GPU architecture can be found in [178], but there are two key features that are relevant for MD codes. Since GPUs were originally designed for carrying out the simple vector operations necessary to render 3D graphics, they are highly efficient for cases when there are many 1000s of identical and independent floating-point operations to be carried out over an array of data. In contrast they are not well suited where this level of *data parallelism* is not available, or where the algorithm requires complex branching and logic. Secondly, the GPU operates as a co-processor with its own independent memory, connected to the main CPU via a communication bus. With GPU memory achieving bandwidth in the 100s of GB/s, communication between the CPU and GPU is much slower, typically 10s of GB/s. As a result code developers have a choice of either moving the entire calculation (including parts which are not well suited to the GPU architecture) to the GPU in order to avoid the communication bottleneck between the CPU and GPU, or make use of the CPU and the GPU in concert, each running separate parts of the calculation, but having to construct complex buffering schemes to minimise the effect of CPU-GPU communication. Codes like AMBER [90, 183] choose the former approach and reach very high efficiency within a single compute node. However, because inter-node communication using MPI requires data to first be moved across the bus from the GPU memory to the CPU memory, across the network to another CPU's memory and finally back over the bus to the GPU this would destroy performance and so is not even implemented. Other codes such as GROMACS [4] and LAMMPS [40, 41] choose the latter approach which typically results in lower performance on an single node but greater maximum performance on large

HPC systems with many GPU-accelerated nodes.

2.2 Related Software Packages

Several software packages exist that either have a similar architecture to MIST, or attempt to address a similar challenge.

PLUMED [205], is similar in design to MIST in that it is a software library that interfaces to a range of MD codes via API calls (which may be inserted using source-code patches). While PLUMED is widely used and at present supports many more MD codes than MIST, it only provides functionality to modify or bias the calculated MD forces and so does not provide read/write access to the atomic positions and velocities. Thus, it is less flexible than MIST and facilitates a much smaller set of MD algorithms. However, because PLUMED restricts itself to a smaller class of algorithms it provides a very straightforward scripting interface, which makes it easy to modify existing algorithms and develop new ones.

A simplified MD program such as NAMD-Lite [93] or MINDY [89] removes much of the complexity of a production MD code, making it easy to modify. However, this results in a loss of functionality (e.g. forcefield support, analysis tools, properties calculations) and performance–restricting the scale and relevance of problems which can be tackled. Many similar simple MD codes exist (e.g. [6, 111, 130, 198]), few having a user-based beyond the immediate research group of the developer. So while it is straightforward to implement new algorithms in these codes, they do little to address the adoption of new methods by the wider MD user community

OpenMM [65] is a toolbox for building MD applications which is designed to be extensible at the source-code level, while being portable to a range of CPU and GPU hardware. The `CustomIntegrator` interface is flexible and provides a Python API to allow declaration of (for example) variables which should be computed for each degree of freedom. However, I argue that this API approach results in code which is less clear and intuitive than the way an integrator is specified in MIST. Similarly, LAMMPS [174] provides an easily extensible object-oriented interface for implementing new integration algorithms. In particular, the concept of a `fix` as a composable object which is applied according to user-specified rules e.g. frequency, group of atoms is very useful for making

modification to existing algorithms. However, one aim of MIST is that the mathematical integration algorithm framework is independent of the MD engine (which takes care of force-field evaluation with periodic boundary conditions, etc.); this portability between codes without having to rewrite any code, enables comparison and cross-validation of results using alternative molecular dynamics codes. Ultimately MIST offers improved interoperability with a broad range of existing codes and force-fields.

The Atomic Simulation Environment (ASE) [129] provides a high-level Python framework for setting up atomistic simulations, running and analysing calculations. It couples to a range of software packages for evaluating forces - both classical and *ab-initio* - so can be used for development of new integrators. While the performance impact of calling between Python and native code at each time step is expected to be slightly (but perhaps not noticably) higher than using MIST, I argue that ASE is aimed primarily at developers or at least code-savvy users who are comfortable with *scripting* their calculations. Similar behaviour is also expected from packages like i-Pi [114] which communicate between the calculation driver and the force evaluation engine over TCP sockets. MIST also provides a high-level interface for integrator developers (in C++ rather than Python), but still retains the familiar input-file driven monolothic execution model that the majority of MD users are more familiar with.

Chapter 3

MIST Library Design and Implementation

In order to address the unmet need for a software package that allows new integrators to be rapidly implemented and tested at scale (see Section 2.1.3) I have developed MIST, the Molecular Integration Simulation Toolkit. As the name suggests, MIST is a software library that provides a set of tools to enable new integration algorithms to be developed for molecular simulations. This chapter describes the key principles that underpin the design of the software, outlines the main architectural elements of the library, and covers in detail relevant implementation details. Of course, software development is an iterative process but this thesis presents the library in its final form. A chronological view of development can be observed in [24–27].

3.1 Design Principles

The MIST library is designed with three key principles in mind:

- **Expressiveness**: MIST must provide a level of abstraction that is sufficiently simple to make writing new algorithms quick and easy, hiding as much as possible the complexity associated with modern MD codes. Developers should be able to write in a syntax that is a close as possible to the mathematics, with as few restrictions as possible on what can be implemented.

- **Performance**: in order to allow algorithms to be tested on relevant-sized problems with realistic force-fields in a reasonable amount of time, MIST must leverage the highly optimised force evaluation routines and parallelisation schemes implemented in mainstream MD packages. This implies that the abstraction that is presented to integrator developers must be general enough to be implemented on top of any arbitrary existing MD code with a low performance overhead.

- **Portability**: to allow for wide uptake of new algorithms, the library must be compatible with a range of established MD software packages. The abstraction presented to the developer should be completely independent of any particular MD code, enabling a write once, run anywhere approach for new integrators. Running a MIST integrator with any 'host' MD code should make use of the same input and output formats, with minimal changes required to select and configure the algorithm.

The extent to which MIST successfully addresses these principles is revisited in section 4.4.

To provide a framework within which these principles can be balanced, I chose to develop MIST as a C++ library. As a compiled language, while the performance benefits over scripting languages such as Python are expected to be relatively small, it makes coupling with existing MD codes (typically written in C, C++ or Fortran) simpler - for example interoperating on shared data structures with low overhead. Some modern codes such as LAMMPS have full Python APIs, but this is by no means the norm. Object orientation makes it straightforward to develop *abstract* representations of the state of the system, with concrete implementations for specific MD codes.

The key components of the MIST library are shown in Figure 3.1 and described in detail in the following sections:

- A library API (Application Programmer's Interface) that facilitates data and control flow transfer between the host code and the MIST library. This API would be used by any developer adding support for MIST into their own MD code.

- A C++ `System` class providing an abstract representation of the state of the system being simulated. This is the interface which integrator developers

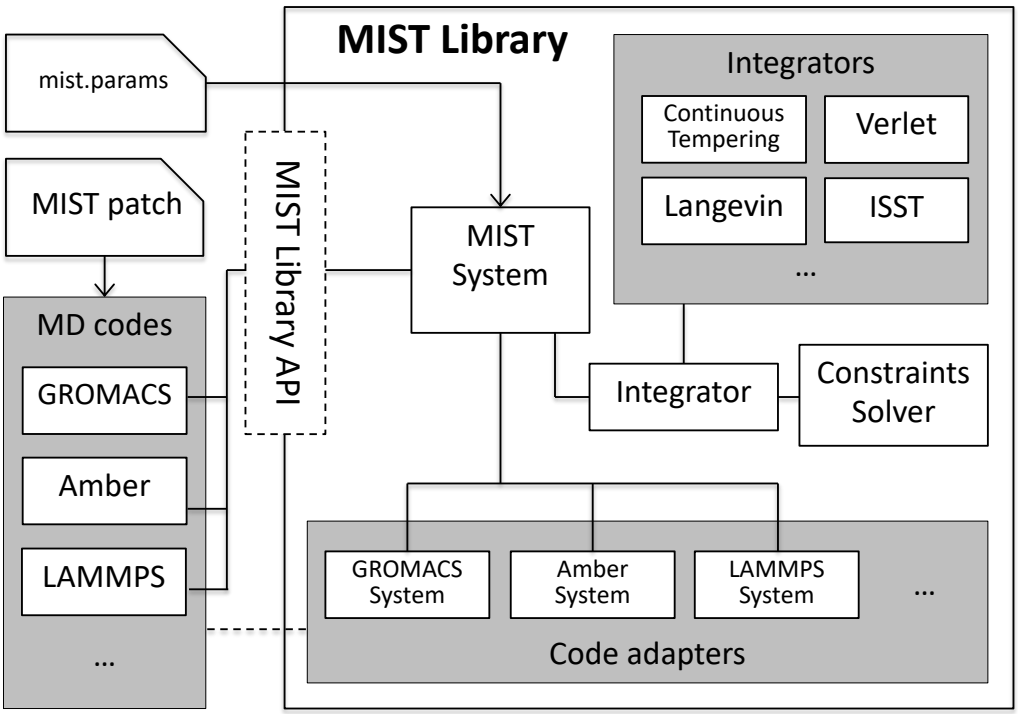


Figure 3.1 *Schematic representation of the main components of the MIST library.*

use to access and modify system state. Adaptor subclasses provide an implementation for each supported MD code, using data pointers registered via the library API to access and modify the system state.

- A set of sample `Integrator`s, which can be selected by the user to perform time integration, independent of the choice of MD code.
- A set of source-code patches or extensions which add MIST library API calls into a set of supported host codes. At time of writing these are LAMMPS, GROMACS, Amber, NAMD-Lite, and Forcite.

3.2 MIST System

The conceptual model of the state of the system is deliberately very simple. A set of n point particles is defined and labeled $0..n-1$, where n is assumed to remain fixed for the duration of the simulation. Each particle has a set of properties: position, velocity, mass, kind (atomic species, typically) and force (the force acting on the particle). In addition, there are several global properties, such as

the cell lattice vectors (if periodic boundary conditions are employed), the total potential energy, and the pressure. These properties fall into three categories: dynamical variables, which may be updated at each timestep by the integrator; read-only data, which are unchanged throughout the simulation; and derived values, which are functions of the other two sets of properties. For example, the particle positions are dynamical variables, the kind of the particle is read-only data, and the force on each particle is a function of the set of particle positions and kinds. All of this state is encapsulated as a C++ `System` class. For the dynamical variables, accessor (e.g. `GetPosition()`) and mutator (e.g. `SetPosition()` methods are provided. For read-only data and derived values only accessors are provided (e.g. `GetForce()` and `GetPressure()`). All real-number variables are represented in double precision through the `System` accessors, irrespective of the precision of the underlying data from the host MD code.

Evaluation of forces is treated as a black-box, and a method `UpdateForces()` is provided to request the forces on each particle to be updated (usually the most expensive operation in an MD simulation). Access to a simple representation of the molecular topology is also provided: a set of b bonds labelled $0..b-1$, where each bond consists of a pair of particle indices and a fixed length, encapsulated as a lightweight `Bond` object.

Most MD codes include some form of parallelisation to improve performance. For shared-memory or GPU parallelisation, all of the particles exist within a single address space. In this case a single instance of the MIST `System` is sufficient to represent the entire state of the system. For distributed memory parallelism, given a set of p processes, it is assumed that the n particles are divided into p subsets of size $n_0..n_{p-1}$, where $\sum_{i=0}^{p-1} n_i = n$. MIST provides two functions to obtain the number of particles: `GetNumParticles()` which returns the number n_i of particles local to the calling process i and can be used e.g. for looping over all local particles, and `GetTotalNumParticles()` which returns the global number of particles n and get be used e.g. for normalisation or degrees-of-freedom calculations. As the simulation progresses and particles move around the simulation space, it is common for MD codes to change the decomposition in order to maintain a good load-balance of work across processes. The only restriction MIST imposes is that the decomposition does not change within the scope of a single call to `MIST_Step`, but may change from one step to the next - in particular when control is passed back to the host code to compute updated forces, the code is guaranteed not to change the decomposition. Thus integrator

developers may assume that the number or ordering of particles does not change within the scope of a single `Step()`, but should not cache any data related to the particles between subsequent calls.

As an abstraction, the `System` provides everything that is needed to implement an integrator (see Section 3.3). However, since the actual data represented by the `System` resides in the host MD code's data structures, a code-specific adaptor is required to implement the MD-code-independent `System` methods using the data structures present in a particular MD code. The choice of MD code is made at compile-time via arguments to the `configure` script used to drive MIST's build process. For simplicity and performance, MIST is provided with access to the raw data structures in an MD code through pointers registered with MIST by the host MD code. This allows the library API (see Section 3.4) to remain completely code-agnostic, and the details of how those pointers are interpreted to yield useful data is encapsulated with the code-specific `System` adaptor classes. For example, GROMACS (by default) stores data as arrays of single-precision floating point whereas Amber and NAMD-Lite use double-precision, and GROMACS stores the inverse masses of particles, rather than the masses themselves. These differences are hidden from the integrator by the `System` abstraction. Differences in units systems are also abstracted, as discussed in Section 3.4.2.

3.3 Integrators

In MIST, an `Integrator` is an abstract class which has a single method which must be implemented by any sub-class: `void Step(double dt)`, as the name suggests, is a function which implements the time integration of the system state from t to $t + dt$ according to some algorithm. To add a new integration algorithms to the library, a developer needs only to create a new class which inherits from `Integrator` and implements the `Step()` method. A number of convenience functions are also included in the base class which simplify coding, for example a velocity Verlet integrator for the NVE ensemble (see Section 3.3.1) is as simple as:

Listing 3.1: Implementation of a velocity Verlet integrator.

```
void VerletIntegrator::Step(double dt)
{
    // Velocity half-step
    VelocityStep(0.5 * dt);

    // Position full step
    PositionStep(dt);

    system->UpdateForces();

    // Velocity half-step
    VelocityStep(0.5 * dt);
}
```

More complex algorithms can be implemented by directly updating individual particle properties using accessor methods of the `System` class. For example, the stochastic part of the Langevin dynamics integrator (Section 3.3.2) is implemented as (`c1` and `c3` are double precision floating-point constants, `v` is a variable of the lightweight `Vector3` type, and `rnd[tid]` is a (thread-local) instance of MIST's random number generator):

Listing 3.2: Example of a more complex velocity update - Langevin Dynamics.

```
for (int i = 0; i < system->GetNumParticles(); i++)
{
    v = system->GetVelocity(i);
    sqrtinvm = system->GetInverseSqrtMass(i); // 1/sqrt(m)
    v.x = c1 * v.x + sqrtinvm * c3 * rnd[tid]->random_gaussian();
    v.y = c1 * v.y + sqrtinvm * c3 * rnd[tid]->random_gaussian();
    v.z = c1 * v.z + sqrtinvm * c3 * rnd[tid]->random_gaussian();
    system->SetVelocity(i, v);
}
```

Some integrators make specific requirements of the host code, which for the sake of efficiency can be turned off if not needed. For example, most MD codes do not compute the potential energy at every step, since only the forces are required for most integration schemes and computing the total energy in a parallel calculation requires an expensive global communication. However, some integrators such as the tempering schemes (see Section 3.3.6) need the potential energy updated at every time step. MIST defines a set of 'feature flags' that are used to signal any special requirements that the integrator might place on the host code, such as `MIST_FEATURE_REQUIRES_ENERGY_WITH_FORCES` for the aforementioned example, or `MIST_FEATURE_FORCE_COMPONENTS` if access to individual components of the force-field is required. This allows integrators to be coded quite generally and functionality be added incrementally to the MD code adaptors, with a check

Table 3.1 *MIST integrator feature flags.*

Flag (MIST_FEATURE_*)	Description
NONE	Default: no special features required.
FORCE_COMPONENTS	The Integrator requires access to individual components of the per-particle forces. MIST defines the total force F acting on a particle as the sum of F_{bond} (2-body bonded term), F_{bend} (3-body angular term), $F_{dihedral}$ (torsional forces), $F_{improper}$ (forces typically used to keep planar molecules flat), and $F_{nonbonded}$ (all other contributions).
POS_VEL_NO_OFFSET	At the start if the timestep all state variables are assumed to be set to their value at time t.
POS_VEL_OFFSET_PLUS_HALF_DT	At the start of the timestep the velocities are assumed to be set to their value at time $t + dt/2$ i.e. half a step ahead of the positions.
REQUIRES_ENERGY_WITH_FORCES	The integrator requires that during a call to UpdateForces(), the potential energy is also updated as it will be read by MIST.
MODIFIES_CELL	The integrator may modify the lattice vectors.
REQUIRES_PRESSURE_WITH_FORCES	The integrator requires that during a call to UpdateForces(), the (scalar) pressure is also updated as it will be read by MIST.

performed at startup to see if the features of the selected integrator are supported by the code. The full list of feature flags are shown in table 3.1.

Integrators may be configured by parameters read from an input file. When an `Integrator` is constructed in MIST, it is passed a `Param` data structure which can be interrogated to obtain any parameters set by the user. If these are not available, the integrator should provide some reasonable defaults. For example, an NPT integrator might default to a temperature of 300K and pressure of 1 Atmosphere. The full list of integrators and possible parameters (including default values and units) is available online at `https://bitbucket.org/extasy-project/mist/wiki/MIST%20Integrators`. Each integrator is described in detail in the following subsections, highlighting both the functionality available to users, but also illustrating implementation details for the benefit of future integrator developers.

3.3.1 Verlet

Verlet's method is probably the most common integration algorithm for Molecular Dynamics and is implemented in almost all major MD codes in some form. It requires only a single force evaluation per timestep, needs no additional storage beyond the state variables being integrated, and has the desirable properties of being both *symplectic* and *time-reversible*. With a discretisation error of $\mathcal{O}(\delta t^2)$ stable integration can be achieved with moderate timesteps (typically 1fs for most atomistic systems) and so is well suited for long MD runs where precise dynamics are desired. As a symplectic integrator of a Hamiltonian system, without any modification to the integrator the total energy is conserved and so Verlet's method samples the microcanonical (NVE) ensemble.

Velocity Verlet

MIST's Velocity Verlet integrator is implemented in the `VerletIntegrator` class and is selected by the `integrator verlet` input keyword. As an NVE integrator, there are no additional parameters that can be set by the user, except for configuration of the constraint solver, if required (see Section 3.3.8).

Verlet Leapfrog

Most integrators advance the entire state of the system from time t to $t+dt$. However, a class of algorithms such as Verlet integration in the 'leapfrog' formulation operate assuming the velocities to be offset by $dt/2$ from the positions at the start of each step. This offers a very small computational saving over the 'velocity' formulation since only a single (full-step) velocity update is required at each timestep rather than two (half-step) updated. In MIST, such an integrator must be labeled with the 'feature flag' `MIST_FEATURE_POS_VEL_OFFSET_PLUS_HALF_DT`, to ensure that account of this is taken by the host MD code (for example, computing the kinetic energy at time t based on averages over the two nearest known velocities at $t - dt/2$ and $t + dt/2$). In practice, since the computational cost of the updates is usually vanishingly small compared to the force evaluation, these complexities are avoided by using the 'velocity' formulation. However, this integrator is included in MIST for validation against codes (including Amber) which use Verlet Leapfrog.

The Verlet Leapfrog integrator is implemented in the `LeapfrogIntegrator` class and is selected by the `integrator leapfrog` input keyword. There are no additional parameters.

3.3.2 Langevin Dynamics

In contrast to Verlet integration which solves the Hamiltonian equations of motion to obtain a dynamical trajectory, if following an exact trajectory is not required because the goal of a calculation is to compute observables via ensemble averages then the problem can be re-cast into a Langevin formulation [136], where the equations of motion for each particle become stochastic, including a dissipative (drag) force, and random perturbative force $\boldsymbol{\eta}$ constructed as shown in Section 3.4.2 to effectively thermostat the system at a particular temperature:

$$m\frac{d^2\boldsymbol{r}_i}{dt^2} = F(\{\boldsymbol{r}\}) - \gamma\frac{d\boldsymbol{r}_i}{dt} + \boldsymbol{\eta}(t)$$

As shown in [132], this formulation has two key advantages - the random perturbation ensures it is *ergodic*, guaranteeing a complete sampling of the phase space and the drag force damps resonances that could destroy the stability of deterministic methods due to damping of resonances which. With a careful construction, this can allow for longer timesteps than the stability limit of Verlet integration $\delta t < \omega/2$ (where ω is the natural frequency of the fastest oscillator in the system) without affecting ensemble averages.

The `LangevinIntegrator` in MIST uses the BAOAB and ABOBA splitting methods which are derived in [131] and shown to give much better performance than a range of other constructions, sometimes referred to as 'Langevin Thermostats'. The integrator is selected by the `integrator langevin` input keyword, and the `temperature` and `langfriction` are exposed to the user as parameters. By default, a seed for the random number generator is chosen based on the current timestamp, however for reproducibility this can also be set by a parameter `seed` if desired.

3.3.3 Nose-Hoover NPT

Among a wide variety of schemes for temperature control, the Nosé-Hoover method [102, 160, 161] is popular for a wide range of systems as it is a deterministic method so preserves 'exact' dynamics without the unphysical behaviour associated with methods which directly rescale velocities such as the Andersen[9] and Berendsen[20]. Nosé-Hoover is an example of an 'extended-system' approach where additional dynamical variables are added to the system. In this case a 'heat bath' at fixed temperature which is coupled to the particles in the system and exchanges energy until the system reaches equilibrium at the desired temperature. While the original method was shown not to be ergodic [175], ergodicity may be recovered by the addition of a chain (typically 3) of interlinked thermostats [147].

The `NoseHooverIntegrator` in MIST provides an example of how to implement an extended system scheme. In addition to temperature control using Nosé-Hoover chains, pressure control is achieved using a 'barostat' along the lines of the Parrinello-Rahman [164, 165] approach. As there are a wide variety of possible implementation choices, for the sake of simplicity while demonstrating the capability of MIST, I implemented a barostat which allows for isotropic expansion or contraction of the simulation cell following the equations of motion described in [190, 207]. In particular, this requires only a scalar pressure rather than a 3-dimensional tensor quantity needed for a fully flexible cell.

Implementation in MIST is straightforward - the integrator declares the `MIST_FEATURE_MODIFIES_CELL` and `MIST_FEATURE_REQUIRES_PRESSURE_WITH_FORCES` flags, to indicate that it will call `SetCell()` and `GetPressure()` respectively. The extended state variables - the thermostat chain, and the barostat and its associated chain - are stored as member variables so they are preserved between timesteps, and the ordering of update steps is as follows. Also shown is the detail of the isotropic rescaling update:

Listing 3.3: Implementation of a Nose-Hoover NPT integrator.

```
void NoseHooverIntegrator::Step(double dt)
{
    // Barostat NH chain half-step & scale barostat velocity
    BarostatNHC(0.5 * dt);

    // Thermostat half-step & scale particle velocity
    Thermostat(0.5 * dt);

    // Barostat half-step
    BarostatVelocityStep(0.5 * dt);
    BarostatRescaleVelocities(0.5 * dt);

    // Velocity half-step
    VelocityStep(0.5 * dt);

    // Box rescale half-step
    RescaleCell(exp(eps_v * 0.5 * dt));

    // Position full step
    PositionStep(dt);

    // Box rescale half-step
    RescaleCell(exp(eps_v * 0.5 * dt));

    system->UpdateForces();

    // Velocity half-step
    VelocityStep(0.5 * dt);

    // Barostat half-step
    BarostatRescaleVelocities(0.5 * dt);
    BarostatVelocityStep(0.5 * dt);

    // Thermostat half-step & scale particle velocity
    Thermostat(0.5 * dt);

    // Barostat NH chain half-step & scale barostat velocity
    BarostatNHC(0.5 * dt);
    }

void NoseHooverIntegrator::RescaleCell(double scale)
{
    Vector3 a, b, c;

    system->GetCell(&a, &b, &c);
    system->SetCell(a*scale, b*scale, c*scale);
}
```

To simplify the implementation of integrators which modify the cell, the `SetCell()` method used to update the lattice vectors performs a rescaling of the particle positions so they retain the same fractional coordinates the new cell. Without loss of generality, the first lattice vector is enforced to lie along the x

axis and the second lattice vector to lie in the xy plane i.e. the box matrix is upper-triangular, in order to simplify integration with MD codes, many of which do not support fully general cells. With the box matrix $\boldsymbol{h}$ defined as the 3x3 matrix formed of the lattice column vectors $\boldsymbol{a}$, $\boldsymbol{b}$ and $\boldsymbol{c}$:

$$\boldsymbol{h} \equiv \begin{pmatrix} a_1 & b_1 & c_1 \\ 0 & b_2 & c_2 \\ 0 & 0 & c_3 \end{pmatrix}$$

Then the cell update simply becomes:

1. Transform particle coordinates to fractional coordinates by $\boldsymbol{r}_i' = \boldsymbol{h}^{-1}\boldsymbol{r}_i$
2. Update lattice vectors and rebuild box matrix
3. Transform particle coordinates into new box by $\boldsymbol{r}_i = \boldsymbol{h}\boldsymbol{r}_i'$

MIST's implementation also includes as an option the modification proposed in [5] to quench cell vibrations during equilibriation of crystalline systems. This modification simply removes kinetic energy from the box vibration by applying the following test at each step (where $\dot{\epsilon}$ is the velocity of the box degree of freedom and $\ddot{\epsilon}$ the corresponding acceleration):

$$\text{If} \quad \dot{\epsilon}.\ddot{\epsilon} < 0 \quad \text{Then} \quad \dot{\epsilon} = 0$$

The effect of this modification on the dynamics of the system is discussed in Section 4.2.

The Nosé-Hoover NPT integrator is selected by the `integrator npt` input keyword. The behaviour of the thermostat can be controlled by the parameters `temperature`, `temp_chain_length` and `temp_nsteps_relax` (which controls the equilibriation timescale of the thermostat by setting the thermostat mass). Similarly, the parameters `pressure`, `press_chain_length` and `press_nsteps_relax` control the barostat. The Ackland box-quench modification is disabled by default and can be enabled by setting `box_quench true`.

3.3.4 Runge-Kutta 4th Order

As discussed in Section 2.1, Runge-Kutta integration schemes are not usually preferred for MD. However, to illustrate that it is possible to implement integrators with MIST which require multiple force-evaluations per timestep I included the `RK4Integrator` following the 'classic' 4th-order algorithm from [125] to sample the NVE ensemble with discretisation error $\mathcal{O}(\delta t^4)$. The implementation is only notable in that the mid-step positions and velocities must be stored as temporary variables, thus the total storage requirements of the integrator are quite large - with $10 \times n_{particles}$ additional double-precision values allocated by MIST. Thus this integrator serves as an additional example of how to manage temporary storage in MIST. Any array temporaries are allocated on the first entry to the `Step()` method and must be deallocated in the class destructor. Nevertheless, the core of the update step is rather simple:

Listing 3.4: Implementation of a Runge-Kutta 4th order integrator.

```
// Store original Positions and Velocities
for (i = 0; i < system->GetNumParticles(); i++)
{
    oldPos[i] = system->GetPosition(i);
    oldVel[i] = system->GetVelocity(i);
}

// Compute k1 and l1 based on initial forces
for (i = 0; i < system->GetNumParticles(); i++)
{
    m_inv = system->GetInverseMass(i);
    k1[i] = system->GetForce(i) * dt * m_inv;
    l1[i] = oldVel[i] * dt;
    system->SetPosition(i, oldPos[i] + l1[i] * 0.5);
}

// Compute k2 and l2 based on first estimate to the forces at t+dt/2
system->UpdateForces();

for (i = 0; i < system->GetNumParticles(); i++)
{
    m_inv = system->GetInverseMass(i);
    k2[i] = system->GetForce(i) * dt * m_inv;
    l2[i] = (oldVel[i] + k1[i] * 0.5) * dt;
    system->SetPosition(i, oldPos[i] + l2[i] * 0.5);
}

// Compute k3 and l3 based on second estimate to the forces at t+dt/2
system->UpdateForces();

for (i = 0; i < system->GetNumParticles(); i++)
{
    m_inv = system->GetInverseMass(i);
    k3[i] = system->GetForce(i) * dt * m_inv;
    l3[i] = (oldVel[i] + k2[i] * 0.5) * dt;
    system->SetPosition(i, oldPos[i] + l3[i]);
}

// Compute k4 and l4 based on estimate to the forces at t+dt
// and set final positions and velocities
system->UpdateForces();

for (i = 0; i < system->GetNumParticles(); i++)
{
    m_inv = system->GetInverseMass(i);
    k4[i] = system->GetForce(i) * dt * m_inv;
    l4[i] = (oldVel[i] + k3[i]) * dt;
    system->SetPosition(
        i, oldPos[i] + (l1[i] + l2[i] * 2 + l3[i] * 2 + l4[i]) * one_sixth);
    system->SetVelocity(
        i, oldVel[i] + (k1[i] + k2[i] * 2 + k3[i] * 2 + k4[i]) * one_sixth);
}

system->UpdateForces();
```

Making the assumption that the force depends only on the particle positions and is not a function of the velocities, there is no need to call `setVelocity()` until the end of the step. Compared to previous integrators, the loop over particles is made explicit rather than using the convenience functions `PositionStep()` and `VelocityStep()` to make it clear when positions are updated in the host code, and which values are temporaries that are only required in the integrator.

The integrator is selected by the `integrator rk4` input keyword and takes no additional parameters.

3.3.5 Yoshida Symplectic Integrators

An alternative approach to constructing integrators for Hamiltonian systems with small errors was derived in [219]. While these schemes require multiple force evaluations per timestep, they have the advantage of being symplectic, leading to long-time stability, and require no additional temporary variables. The fourth-order scheme (discretisation error $\mathcal{O}(\delta t^4)$), is easily implemented as a series of three Verlet-like steps with scaled δt:

Listing 3.5: Implementation of a Yoshida 4th order integrator.

```
double d[3] = {1.351207192, -1.702414384, 1.351207192};

for (int j = 0; j < 3; j++)
{
    // Velocity half-step
    VelocityStep(0.5 * dt * d[j]);

    // Position full step
    PositionStep(dt * d[j]);

    system->UpdateForces();

    // Velocity half-step
    VelocityStep(0.5 * dt * d[j]);
}
```

The integrator is selected by the `integrator yoshida4` input keyword and takes no additional parameters as it samples the NVE ensemble.

As with the 4th-order scheme (Listing 3.5), an integrator with discretisation error $\mathcal{O}(\delta t^8)$ can be constructed from fifteen sub-steps, using the parameters given in [219]. This integrator is selected by the `integrator yoshida8` input keyword and likewise, takes no additional parameters.

3.3.6 Advanced Sampling Algorithms

As discussed in Section 2.1.3, in order to rapidly sample the phase-space of complex systems a range of schemes have been developed to enable simulations to more rapidly escape from one local minima of the potential energy surface, while still retaining the ability to calculate ensemble averages. These typically either modify the force-field (e.g. Metadynamics [128] or Accelerated Molecular Dynamics [91]) to reduce barrier heights or 'push' the system away from already explored regions of space, or increase the rate of exploration by increasing the temperature of (parts of) the system, giving enough energy to cross barriers. A range of such algorithms in MIST are implemented in MIST, some of which have been developed by collaborators, and others which are adaptions of previously published algorithms.

Continuous Tempering

Continuous Tempering is an extended-variable scheme developed and implemented independently in MIST by Gobbo and Leimkuhler [81] and which I subsequently refactored and improved, including adding parallelisation. Briefly, the algorithm introduces an additional degree of freedom ξ coupled to the potential energy by a coupling function $f(\xi)$:

$$
\begin{aligned}
&f(\xi) = 0, |\xi| < \Delta, \\
&f(\xi) = S_f \left[3 \left(\frac{|\xi| - \Delta}{\Delta' - \Delta} \right)^2 - 2 \left(\frac{|\xi| - \Delta}{\Delta' - \Delta} \right)^3 \right], \Delta < |\xi| < \Delta, \\
&f(\xi) = S_f, |\xi| > \Delta'
\end{aligned}
$$

where S_f is chosen so as to set a maximum effective temperature.

Metadynamics is used to achieve a uniformly distributed sampling of ξ. Depending on the choice of Δ and Δ', the system spends approximately 1/3 of the time in the unperturbed region $|\xi| < \Delta$ where $f(\xi) = 0$ and the unperturbed system behaviour is recovered. Configurations selected from this sub-set are representative of the canonical ensemble and can be used without having to post-process the simulation results with any complex reweighting schemes to obtain

free energies and other observables at the physical temperature.

The `ContinuousTempering` implementation in MIST is closely based on the Langevin BAOAB integrator (Section 3.3.2), and indeed is a sub-class of the `LangevinIntegrator` class to avoid code duplication. In addition to the parameters inherited from the parent class, there are parameters controlling both the coupling function (`temp_fact`, `delta` and `delta2`) and the dynamics of the extended variables (`hills-height`, `hills-width`, `metadyn_pace`, `langfriction_xi` and `mass_xi`). The integrator is selected using the input keyword `integrator tempering`.

TAMD

This integrator, developed by Ralf Banisch is closely based on the extended-variable formulation of Temperature-Accelerated MD [144] described in [51]. The implementation is designed for sampling the free energy surface of a biomolecular system such as alanine dipeptide, where the free energy surface known to be well described by two collective variables associated with the 'Ramachandran angles' Φ and Ψ - the two backbone dihedral angles either side of the central carbon atom. Two additional degrees of freedom are defined with large associated masses and a high (TAMD) temperature which are coupled to the dihedral angles themselves by stiff harmonic springs. Unlike the original versions of the algorithm which used deterministic integrators, here the dynamics of both the molecular system and the extended system are propagated using the Langevin BAOAB scheme (Section 3.3.2) at their respective temperatures. So the order of updates is as follows:

1. Half step particle velocities (B)
2. Correction to particle velocities due to additional DOFs
3. Half step extended system velocities (B)
4. Half step particle positions (A)
5. Half step extended system positions (A)
6. Apply drag and perturbation to particle velocities at system temperature (O)

7. Apply drag and perturbation to extended system velocities at TAMD temperature (O)
8. Half step particle positions (A)
9. Half step extended system positions (A)
10. Update forces on particles
11. Calculate forces on extended system
12. Half step particle velocities (B)
13. Correction to particle velocities due to additional DOFs
14. Half step extended system velocities (B)

The `TAMD` integrator is selected using the input keyword `integrator tamd`. In addition to parameters inherited from the `LangevinIntegrator`, the integrator requires a corresponding pair of `tamd_temp` and `tamd_friction` parameters to control the integration of the extended system, `tamd_mass` and `tamd_kappa` define the mass of the extended system variables and the strength of the coupling respectively and a list of indices `phi1`...`phi4` and `psi1`...`psi4` define the sets of particles which are used to compute the dihedral angles. The forces on the extended variables are written to file every `tamd_save_freq` for post-processing.

Continuous Tempering + TAMD

One of the goals of MIST was to make it easy to rapidly develop and test new algorithms. This scheme, developed by Ralf Banisch, is a novel combination of the previous two methods. There are a total of three addition variables introduced - the two coupled to Φ and Ψ (following the TAMD implementation) and the ξ from Continuous Tempering. Continuous Tempering is applied to the dihedral degrees of freedom only, with the coupling function $f(\xi)$ acting as a scaling factor. The sequence of updates at each timestep is formed by the interleaving of the Continuous Tempering updates with those from TAMD as shown (modifications to TAMD shown *in italic*):

1. Half step particle velocities (B)

2. Correction to particle velocities due to additional DOFs
3. Half step Φ and Ψ velocities *(scaled by $f(\xi)$* (B)
4. *Half step ξ velocity (B)*
5. Half step particle positions (A)
6. Half step Φ and Ψ positions (A)
7. *Half step ξ position (A)*
8. Apply drag and perturbation to particle velocities at system temperature (O)
9. Apply drag and perturbation to Φ and Ψ velocities at TAMD temperature (O)
10. *Apply drag and perturbation to ξ velocity at system temperature (O)*
11. Half step particle positions (A)
12. Half step Φ and Ψ positions (A)
13. *Half step ξ position (A)*
14. Update Forces on particles
15. Calculate forces on Φ and Ψ
16. *Calculate force on ξ*
17. Half step particle velocities (B)
18. Correction to particle velocities due to additional DOFs
19. Half step Φ and Ψ velocities *(scaled by $f(\xi)$* (B)
20. *Half step ξ velocity (B)*

The `ContinuousTempering_TAMD` integrator is selected by the `integrator tempering_tamd` input keyword. The integrator accepts the combined set of parameters from both `ContinuousTempering` and `TAMD`.

Simulated Tempering

A more classic approach to sampling the free energy landscape of complex system is the Simulated Tempering algorithm of Nguyen *et al* [159, 221]. Previously this has only been made available as a set of shell scripts [220] for use with GROMACS.

Simulated Tempering consists of selecting a set of states with discrete temperatures $T_1 < T_2 < ... < T_n$ ranging from the target temperature of interest up to a maximum chosen to permit the system enough kinetic energy to cross any relevant energetic barriers, with associated weights $f_1 ... f_n$. Starting in one of the states, say T_i, the system is simulated for a set `period` (1000 timesteps, by default) under the control of thermostat, and the average potential energy observed in the state is recorded. To avoid accumulating rounding errors we adjust the average potential energy $\bar{U}_i$ in state i at each step by:

$$\bar{U}_i := (U - \bar{U}_i)/n_i$$

where U is the instantaneous potential energy and n_i is the number of samples (timesteps) accumulated in that state.

Every `period` steps, the weights are updated using the following recurrence:

$$f_i = f_{i-1} + (\beta_i - \beta_{i-1}) \times (\bar{U}_i + \bar{U}_{i-1})/2$$

where β_i is the Boltzmann factor $1/k_B T_i$. The system then attempts a transition from the current state to a neighbouring state with probability:

$$p_{a \to b} = \exp[(\beta_a - \beta_b) \times U - (f_a - f_b)]$$

As the simulation progresses, the weights are iteratively adjusted until they reach an equilibrium where the transition probabilities to neighbouring states are equal ($p_{i \to i-1} = p_{i \to i+1}$). The system will then make a random walk among the temperature states, and the free energy or other observable at any temperature of interest can be extracted either by picking out only the configurations generated at that temperature, or by rescaling using a method like WHAM [123] or similar reviewed in [218]. The only technicality is to set the temperature states close

enough together relative to the potential energy fluctuations of the system that p is not vanishingly small and the system remains stuck in a single state.

To set up a simulation using the previously published scripts required creating separate GROMACS input files for each temperature state, then running multiple short simulations, where the potential energy is parsed from the output file and a probabilistic change to another temperature state is made according to the algorithm. As a result, the scripts generate a set of trajectory data files, which must be concatenated for analysis, and running many short individual simulations makes it inefficient to operate through an HPC batch system.

An additional novel element of the implementation is that the temperature of each state is controlled using Langevin Dynamics implemented using a BAOAB splitting scheme [132] to give more accurate configurational averages. The simulations discussed in see Section 3.3.6 are not designed to test this assertion, but a thorough analysis by Fass *et al* [69] showed dramatic reduction in configuration space discretisation error compared with other schemes.

Running Simulated Tempering through MIST needs only a single long MD run. The Simulated Tempering algorithm is selected and configured by a `mist.params` file as follows:

```
integrator simulated_tempering # Select Simulated Tempering
temperatures[0] 300            # Define a series of temperature states
temperatures[1] 310
temperatures[2] 320
...
temperatures[14] 440
temperatures[15] 450
period 2500                    # Attempt to switch states every 2500 steps
constraints all-bonds          # Apply bond constraints

langtemp 300                   # Start system at 300K
langfriction 1.0               # 1/ps friction constant
```

Infinite Switch Simulated Tempering

This integrator was developed and implemented by Martinsson *et al* and its derivation and properties are described in [146], where my assistance with the implementation is acknowledged. Briefly, Infinite Switch Simulated Tempering (ISST) is the extension of the Simulated Tempering method where the set of discrete temperature states are replaced by a continuously varying temperature, in the limit where the period between temperature state switches $\tau \rightarrow 0$ - the infinite switch limit. Rather than computing discrete weights for each temperature state as in standard Simulated Tempering, the weight is treated as a continuous function and approximated by a Legendre polynomial. The ordering of updates is based on the Langevin BAOAB scheme, adapted as follows:

1. Half step particle velocities (B) *with rescaled forces*
2. Half step particle positions (A)
3. Apply drag and perturbation to particle velocities at system temperature (O)
4. Half step particle positions (A)
5. Update Forces on particles
6. *Update approximation to the weight function*
7. *Recompute the force rescaling factor per equation 58 in [146]*
8. Half step particle velocities (B) *with rescaled forces*

One of the key advantages of ISST is that it has many fewer parameters than standard Simulated Tempering. Besides the temperature limits `temperature_min` and `temperature_max` the only choice required is the number of points `npts_interpolation` used in the approximation to the weight function and a scaling factor `learn_scaling` which controls the rate at which the weight function is adjusted. The implementation inherits from `LangevinIntegrator`, so any parameters understood by that integrator are also accepted. The `ISST` integrator is selected by the `integrator ISST` input keyword.

3.3.7 Parallelisation

In order to achieve performance on a par with production MD codes, MIST supports both shared-memory (OpenMP) and distributed-memory (MPI) parallelisation, and may make use of either (or both) to match the configuration of the host code.

Since OpenMP support is included in all major compilers, and the OpenMP directives in the source code are ignored otherwise, OpenMP support is enabled by default in MIST. If a MIST-enabled MD program is run with multiple OpenMP threads (typically by specifying the `OMP_NUM_THREADS` environment variable, MIST may spawn threads to speed up any key computational bottlenecks in the integrator. Loop-level parallelism can be added explicitly, for example:

Listing 3.6: Explicit OpenMP loop-level parallelism for a velocity rescaling step

```
#pragma omp parallel for default(none) private(v) shared(n, scale)
  for (int i = 0; i < n; i++)
  {
    v = system->GetVelocity(i);
    system->SetVelocity(i, Vector3::Scale(scale, v));
  }
```

A number of common operations such as velocity and position time steps that are implemented in the base `Integrator` class contain OpenMP statements embedded in them so subclasses can take advantage of improved performance without the need to add OpenMP directly. The same is true of the `ConstraintSolver` functionality described in Section 3.3.8. As usual in an OpenMP program, it is up to programmer to ensure that any access to shared state such as class variables is done in such a way to avoid race conditions. An example of this can be found in the `LangevinIntegrator` class, where each thread has its own random number generator used in the stochastic 'O' step:

Listing 3.7: Example of thread-safe random number generation from `LangevinIntegrator`

```
// Array of MIST_Random objects
MIST_Random **r;

...

// Each RNG is initialised and seeded

int nthreads = 1;
#ifdef _OPENMP
nthreads = omp_get_max_threads();
#endif
r = new MIST_Random *[nthreads];
for (int i = 0; i < nthreads; i++)
{
  r[i] = new MIST_Random(seed + i);
}

...

// Each thread accesses its own RNG r[t]

int t = 0;
#pragma omp parallel default(none) private(v, t, sqrtinvm) shared(c1, c3)
{
#ifdef _OPENMP
  t = omp_get_thread_num();
#endif
#pragma omp for
  for (int i = 0; i < system->GetNumParticles(); i++)
    {
      v = system->GetVelocity(i);
      sqrtinvm = system->GetInverseSqrtMass(i);
      v.x = c1 * v.x + sqrtinvm * c3 * r[t]->random_gaussian();
      v.y = c1 * v.y + sqrtinvm * c3 * r[t]->random_gaussian();
      v.z = c1 * v.z + sqrtinvm * c3 * r[t]->random_gaussian();
      system->SetVelocity(i, v);
    }
  }
}
```

The `random_gaussian()` function uses the standard Box-Muller [38] approach to generate normally-distributed random numbers from uniformly-distributed real numbers in $[0, 1)$. While this is slightly more computationally costly than using the unformly-distributed random numbers directly, it makes little practical difference as the central limit theorem applies if the number of samples is large, which is the case for any practical calculation.

MPI support in MIST is enabled at compilation time by the `--enable-mpi` argument to the configure script (see Section 3.5). In this case the `MIST_Init_MPI()`

API function is compiled in to the library and once it is called by the host code an MPI communicator handle is passed from the host into MIST which can be used to manage communication between processes. Any code which is required only in the MPI-parallel case can be conditionally compiled using the `__MIST_WITH_MPI` macro as shown in the example below. As described in Section 3.2, each process has its own discrete set of particles which it is responsible for integrating so there only a few cases that an integrator developer has to take particular care over:

1. Any calls to `GetNumParticles()` returns the number of particles local to that process, so any calculations which require the global total number of particles such as calculating the number of degrees of freedom should use `GetTotalNumParticles()` instead.

2. Any calculations of global quantities such as the total kinetic energy should be done by having each process calculating its local contribution and summing these using `MPI_AllReduce()` as shown:

   Listing 3.8: MPI global reduction to calculate the total kinetic energy, taken from the `NoseHooverIntegrator`.

   ```
   // Compute current kinetic energy
   double ke = 0.0;
   Vector3 v;

   // Local sum first
   for (i = 0; i < system->GetNumParticles(); i++)
   {
     v = system->GetVelocity(i);
     ke += Vector3::Mult(v, v) * system->GetMass(i);
   }

   // Optional global sum if MPI is enabled
   #ifdef __MIST_WITH_MPI
   // Sum up to get global KE
   double local_ke = ke;
   MPI_Allreduce(&local_ke, &ke, 1, MPI_DOUBLE, MPI_SUM,
                 system->GetCommunicator());
   #endif

   // Resulting total KE is in the variable ke
   ```

3. Probabilistic choices such as the transition between temperature states in Simulated Tempering need to be agreed between all processes to ensure all parts of the system are maintained in an NVT ensemble at the *same* temperature. The easiest way to achieve this is by simply broadcasting the result from the master process (MPI rank 0) to all the other processes as

shown:

Listing 3.9: MPI broadcast of the temperature shift, taken from `SimulatedTempering`.

```
// Decide which direction to attempt a shift
double x = r[0]->random_uniform();
int shift = 0;
if (x >= 0.5 && state + 1 < n_temperatures)
{
  // Shift up a temp, if possible
  shift = 1;
}
if (x < 0.5 && state > 0)
{
  // Shift down a temp, if possible
  shift = -1;
}

#ifdef __MIST_WITH_MPI
// Master decides which direction to shift
MPI_Bcast(&shift, 1, MPI_INT, 0, system->GetCommunicator());
#endif
```

4. To support load balancing by the host code, MIST allows the number of particles n_p assigned to each process to be changed between calls to the integrator's `Step()` function, although it is guaranteed not to change during any calls to `UpdateForces()`. If the integrator allocates temporary arrays or other variables which depend on the number of particles it must be sure to check that enough space is allocated if the number of particles assigned to that process increases. For example, the `RK4` integrator (Section 3.3.4) has to allocate temporary arrays to store the mid-step positions and velocities. To avoid reallocating these at each timestep, with an associated performance cost they are simply kept from one iterator to another, with a check at the start of the step to ensure the arrays have enough space, resizing them if not:

Listing 3.10: Resizing temporary arrays after the number of local particles increases, adapted from `RK4`.

```
void RK4Integrator::Step(double dt)
{
  // Get the current number of particles on this process.
  int n = system->GetNumParticles();

  // Resize storage arrays if needed
  if (n > arrLen) // arrLen is a class variable persisted from one step to the next.
  {
    if (oldPos != NULL)
    {
        delete[] oldPos;
    }
    ...

    oldPos = new Vector3[n];

    ...

    arrLen = n;
  }

...
```

3.3.8 Constraint Solver

In addition to the force-field i.e. the potential function of the atomic positions $U(\{\boldsymbol{r}\})$, from which the forces and therefore dynamics are derived, it is common in molecular simulations to apply *constraints* to the system. Bonds between light atoms such as hydrogen have a high natural vibrational frequency and when using common integration algorithms these vibrations severely limit the time step which may be used for stable MD (see [135, Chapter 4.2] for a more detailed discussion). For example, the Velocity Verlet method for a harmonic oscillator with frequency Ω has a stability threshold of $\delta t < 2/\Omega$. For applications such as conformational sampling, constraints are typically used to remove such vibrational degrees of freedom from the simulation entirely, for example replacing flexible covalent bonds which are modelled as harmonic springs with rigid (fixed-length) 'rods', thus allowing a larger time step and longer overall simulated time scales to be accessed for the same computational cost. More complex constraints are also possible, including angular (fixing the internal angle between three atoms) and dihedral (fixing the torsional angle defined by four atoms), but these are not currently implemented in MIST.

For integrators to be practically useful, they must be able to generate a series of positions which satisfy the constraints, and so to avoid complicated coordinate transformations, additional steps are needed after the standard time-propagation of the positions and velocities to correct these back onto the *constraint manifold* (the multidimensional surface made up of those points which satisfy the constraints). These functions are provided by the MIST `ConstraintSolver` class and may be called by `Integrator`s.

As described in Section 3.2, MIST has a representation of the molecular topology consisting of a set of bonds which link pairs of atoms (a, b), with an equilibrium bond length l (usually at the minimum of the bond-potential between the two atoms). MIST supports applying constraints to three different groupings of bonds: none (`constraints off`), only bonds involving hydrogen atoms (`constraints h-bonds-only`), and all bonds (`constraints all-bonds`). For the selected set of bonds, the `ConstraintSolver` sets up a list of k *holonomic* constraints (i.e. constraints depending only on the particle positions, and time), between atoms ka, kb of the form:

$$\sigma_k := ||\boldsymbol{r}_{ka} - \boldsymbol{r}_{kb}||^2 - l_k^2 = 0$$

Following the standard approach [182] of considering the force $\boldsymbol{G}_i$ due to each constraint involving a particle i, which is defined by method of Lagrange multipliers as:

$$\boldsymbol{G}_i = -\sum_k \lambda_k \nabla_i \sigma_k$$

Then the constraints can be resolved (up to a defined tolerance) by solving for the Lagrange multipliers λ_k and applying a correction to the unconstrained updated positions $\hat{\boldsymbol{r}}$:

$$\boldsymbol{r}_i(t + \delta t) = \hat{\boldsymbol{r}}_i(t + \delta t) + \sum_k \lambda_k \frac{\partial \sigma_k}{\partial \boldsymbol{r}_i}$$

By solving a second time for the set of Lagrange multipliers μ_k which satisfy the time derivative of the constraints:

$$\frac{d\sigma_k(t)}{dt} = (\boldsymbol{v}_{ka} - \boldsymbol{v}_{kb})(\boldsymbol{r}_{ka} - \boldsymbol{r}_{kb}) = 0,$$

where $\boldsymbol{v} = \dot{\boldsymbol{r}}$.

The unconstrained velocities $\hat{\boldsymbol{v}}$ may then be corrected by:

$$\boldsymbol{v}_i(t + \delta t) = \hat{\boldsymbol{v}}_i(t + \delta t) + \sum_k \mu_k \frac{\partial \sigma_k}{\partial \boldsymbol{r}_i}$$

Iterating through the constraints and adjusting the Lagrange multipliers, results in the RATTLE algorithm [10], and is selected with the keyword `constraints_method rattle`.

MIST also implements the adaptive Symmetric Newton Iteration (SNIP) scheme [19], where a symmetric gradient matrix based on the configurations at the start of the timestep is constructed:

$$\hat{\boldsymbol{R}} \equiv \sigma'(\{\boldsymbol{r}\})\boldsymbol{M}^{-1}\sigma'(\{\boldsymbol{r}\})^t$$

Where $\sigma'(\{\boldsymbol{r}\})$ is the matrix of partial derivatives of the constraints with respect to the atomic coordinates and $\boldsymbol{M}$ is the diagonal matrix of particle masses. Since the definition of a bond constraint involves only a pair of atomic positions, the gradient matrix is sparse, with entries on the diagonal:

$$\hat{\boldsymbol{R}}_{i,i} = ||\boldsymbol{r}_{ia} - \boldsymbol{r}_{ib}||^2 \left(\frac{1}{m_{ia}} + \frac{1}{m_{ib}}\right) = l_i^2 \left(\frac{1}{m_{ia}} + \frac{1}{m_{ib}}\right)$$

And off-diagonal for a pair of bonds i, j where $ia = ja$ (and equivalent expressions for other combinations):

$$\hat{\boldsymbol{R}}_{i,j} = \frac{(\boldsymbol{r}_{ia} - \boldsymbol{r}_{ib})(\boldsymbol{r}_{ia} - \boldsymbol{r}_{jb})}{m_{ia}}$$

MIST can then solve for the set of Lagrange multipliers:

$$\lambda_k(t) = \hat{\boldsymbol{R}}^{-1}\sigma_k(t)$$

Update the positions:

$$\boldsymbol{r}_i(t+\delta t) = \hat{\boldsymbol{r}}_i(t+\delta t) + \sum_k \lambda_k \frac{\partial \sigma_k}{\partial \boldsymbol{r}_i}$$

And iterate these two steps until convergence.

The velocity update is even simpler. As in RATTLE the set of Lagrange multipliers which satisfy the time derivatives of the constraints are solved for directly:

$$\mu_k(t) = \hat{\boldsymbol{R}}^{-1}\sigma_k'(t)$$

And finally, velocities are set as for RATTLE.

Importantly, for this method, the sparsity structure and the diagonal entries of the matrix $\hat{\boldsymbol{R}}$ are fixed for the duration of the simulation (since they depend only on the molecular topology), and the off-diagonal entries are fixed while the constraints are iterated. In MIST, the Eigen library [85] is used to store the sparse matrix, perform a Cholesky factorisation, and solve for the Lagrange multipliers. Eigen is particularly useful since it can perform a symbolic decomposition of the matrix once at the start of the simulation which makes the subsequent factorisation faster. SNIP is the default constraint solution method in MIST. The original solver class was written by Ralf Banisch, and I refactored it for performance and the addition of OpenMP, almost entirely re-writing it in the process. At time of writing, the constraint solver is only available for single-process (shared memory parallelisation only) runs. Experimental MPI-parallel implementations have been investigated, based on Eigen with explicit MPI communication, and the MPI-parallel PETSc [16–18] library - however some issues remain so these are not yet merged into the release version of the library. They are available on branches `Eigen_MPI` and `KSP` at `https://bitbucket.org/extasy-project/mist/branches/`. Attempting to run MIST with constraints enabled in an MPI-parallel calculation will log a warning, and turn the constraints off.

Constraints have been added to a selection of integrators in MIST, as shown in Table 3.2. Using the `ConstraintSolver` class methods `ResolvePositionConstraints()` and `ResolveVelocityConstraints()`, it is straightforward to add support to any integrator, based on the implemented examples.

Table 3.2 *Constraint support in MIST integrators.*

Keyword	Supports Constraints
`verlet`	Yes
`leapfrog`	Yes
`nose-hoover`	Yes
`langevin`	Yes
`RK4`	No
`yoshida4`	No
`yoshida8`	No
`tempering`	Yes
`tamd`	No
`tempering_tamd`	No
`simulated_tempering`	Yes
`ISST`	Yes

3.4 MIST Library API

To enable the linkage between a 'host' MD code and the integrators implemented in MIST in a portable fashion, MIST provides simple C and Fortran 90 APIs. These are designed to be general enough to interface to a wide range of possible MD codes, while allowing just enough data and control to be passed to MIST to implement the abstract `System` interface described in Section 3.2. The C interface is declared in a header `mist.h` and the Fortran interface in a module `mist_f90` which can be included by the host code. The Fortran interface provides the same functionality as its C counterpart, with the only difference being that the functions are name `MIST_F_*` rather than `MIST_*`. For the sake of clarity all the API calls shown below are included in their C variant, although exactly the same principles apply to the use of the Fortran interface. As well as function declarations, a range of predefined constants (the aforementioned feature flags, and error codes) are also part of the interface.

The Fortran 90 API is implemented using the `ISO_C_BINDING` feature of the Fortran standard to provide a thin wrapper over the C API. For example, `MIST_Set_NumParticles()` C API is wrapped as shown:

Listing 3.11: Example of Fortran 90 API wrapper.

```
module mist_f90

use iso_c_binding

...

!! Declaration of an ISO_C_BINDING interface to the C API function which takes
!! an integer argument and returns an integer error code
interface
  integer(C_INT) function MIST_SetNumParticles(n) bind(c,name='MIST_SetNumParticles')
    use iso_c_binding
    integer(C_INT), VALUE :: n
  end function MIST_SetNumParticles
end interface

...

contains

...

!! Fortran module function which calls the C API
integer function MIST_F_SetNumParticles(n) result (ierr)
  integer, intent(in) :: n

  ierr = MIST_SetNumParticles(n)
end function

...

end module mist_f90
```

To add support for a new MD code to MIST, API calls must be inserted in to the host MD code as described below. For the specific set of code versions that are supported, MIST provides source-code patches which are automatically applied to insert the API calls during the build process as explained in Section 3.5 or forked versions of the source code that contain MIST API calls directly.

To simplify the API, MIST is designed as a stateful library and is responsible for its own memory management. The client calling the API always interacts with the same instance of the library, rather than for example having to pass an opaque handle back and forwards with each call. Full documentation of the APIs are provided in Doxygen format with the source code, but the key concepts and ordering of API calls are shown in Figure 3.2, which outlines the typical control flow between a host MD code and MIST. Detailed descriptions of how the API calls are integrated into each supported MD code are given in Section 3.4.3.

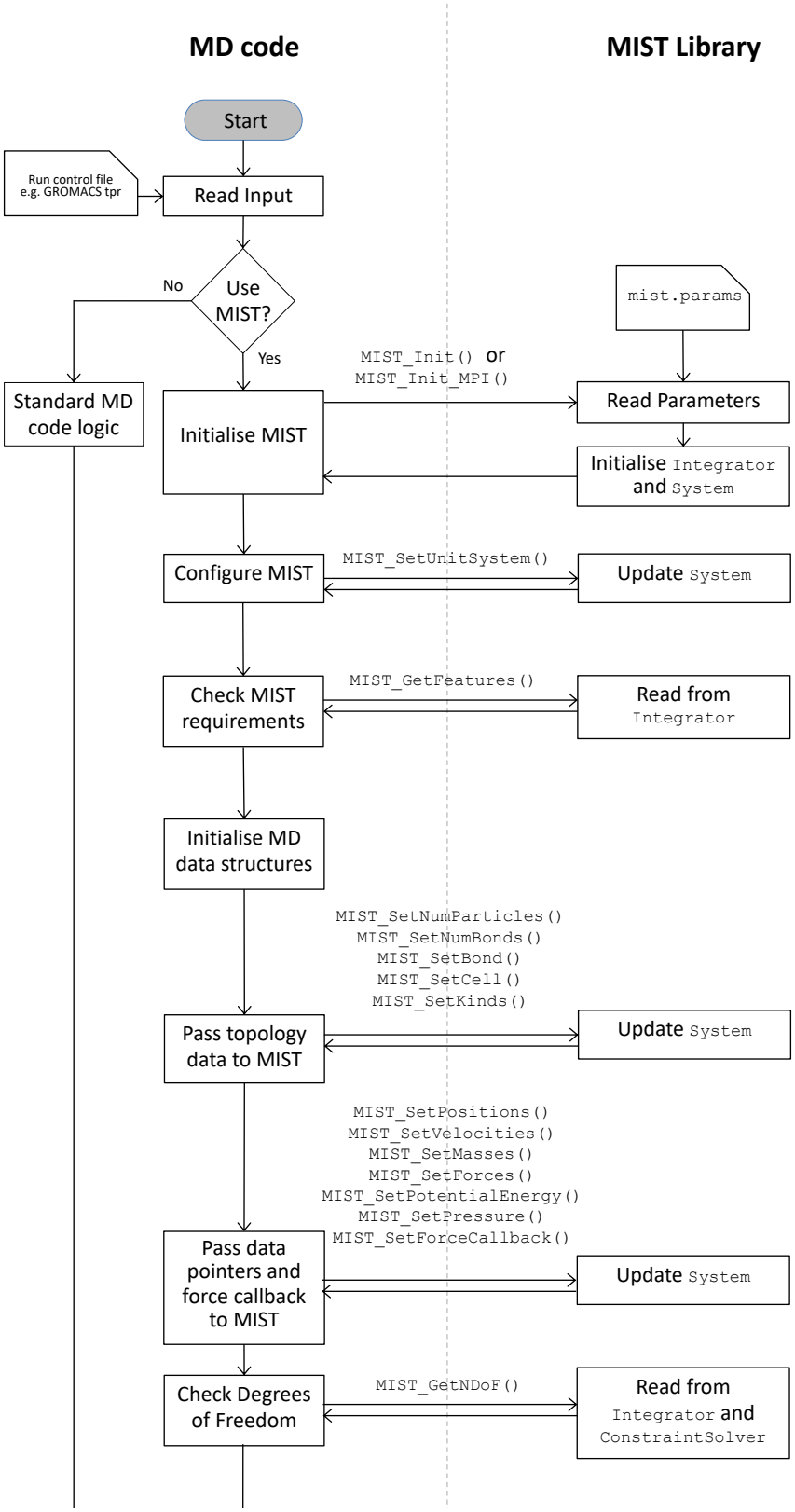
It is important to note that by design, MIST extends the existing functionality of an MD code, and so if MIST is turned off - either as an option in the code's input configuration, at compile-time or by any other method - the MD code should behave exactly as normal.

3.4.1 Control Flow

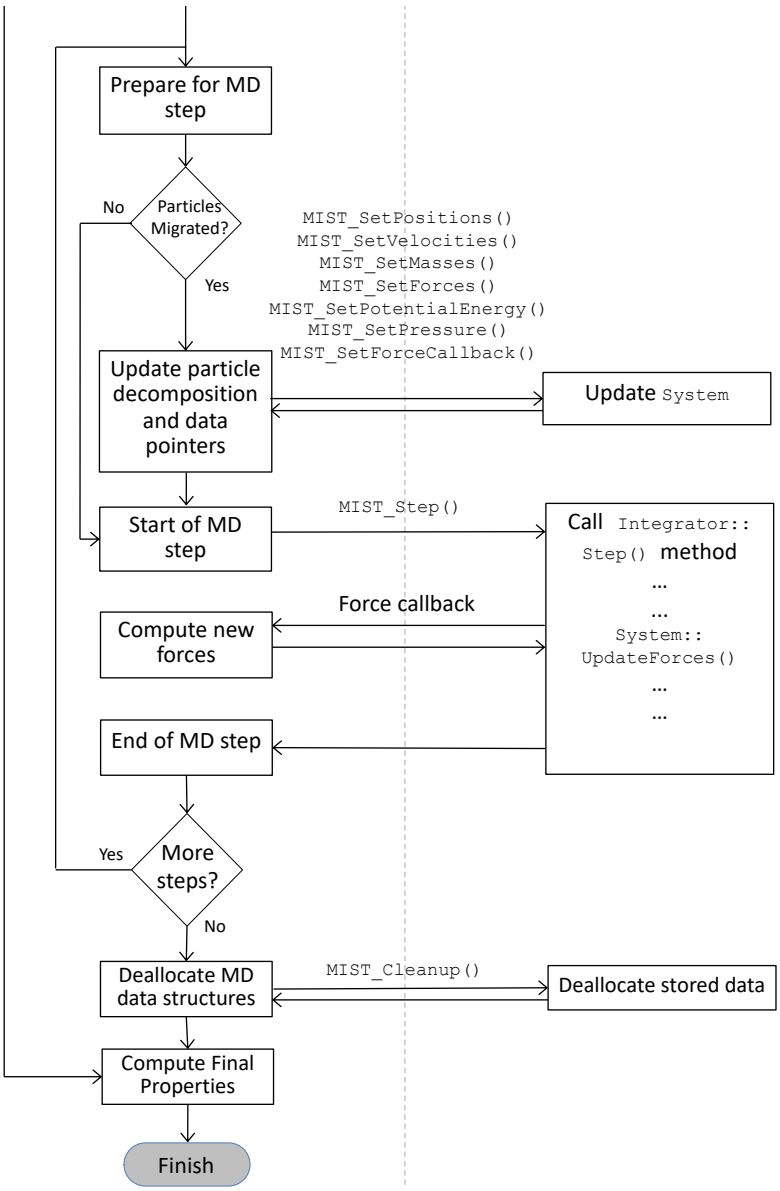
Assuming that MIST has been enabled, the first step is to initialise the library by calling `MIST_Init()` (or `MIST_Init_MPI()` for MPI parallel runs). This triggers MIST to read its own `mist.params` input file and initialise the `System` and `Integrator` objects accordingly.

Once the host code has completed initialisation to the point where the molecular topology is built and initial coordinates and velocities for the particles are assigned, this data must be passed to MIST. A series of calls to (for example) `MIST_SetNumParticles()`, `MIST_SetNumBonds()` and `MIST_SetPositions()` are used to inform MIST of the number of atoms and bonds, and to pass a pointer to the location of the particle position data. In order to be completely flexible, data is passed through the API using void pointers, which are interpreted by MIST in the code-specific adaptor classes to yield data in the standard internal format provided by the `System` class for use by integrators. Detail of how this is used by each supported code is given in Section 3.4.3. The design decision to store pointers to the host data, enabling the library to directly modify the simulation state, is chosen because it is more efficient (reduced memory footprint, memory bandwidth and operations) than the alternative of making an MIST-internal copy of the data, modifying that and explicitly copying it back when required e.g. before force updates, or the end of the MD step.

In addition to passing data pointers into MIST, the host MD code must also register a force callback function pointer and associated parameters by calling `MIST_SetForceCallback()`. This callback function may be called by MIST during an MD step as a black-box to compute updated forces given the current atomic positions (and in general velocities, although none of the supported codes have velocity-dependent forces). Once again, MIST uses a fully generic callback prototype, which accepts a single void pointer for any input data which may be required. Since the actual force computation routine typically does not conform to this interface, it is convenient to define a lightweight parameter data type to store all the arguments which should be passed to the force computation routine



(a) *Initialisation and setup*



(b) *MD step and cleanup*

Figure 3.2 *Control flow in an MD code using the MIST library.*

and a wrapper function which unpacks the type and calls the appropriate function to compute updated forces. For example, in GROMACS:

Listing 3.12: Example of a force callback wrapper.

```
// Declaration of a data type which contains all of the context
// used by the GROMACS force routines
typedef struct {
  FILE *log;
  t_commrec *cr;
  ...
  int flags;
  force_arrays_t *forces;
} force_params_t;

// The wrapper function which takes the context as a void pointer,
// unpacks all of the parameters, and calls the force routine.
void do_force_wrapper(void *params){
    force_params_t *p = (force_params_t *)params;
    do_force(p->log,p->cr,p->inputrec,*(p->step),p->nrnb, \\
             *(p->wcycle),p->top,p->groups,*(p->box),p->x, \\
             p->hist,p->forces,*(p->vir_force),p->mdatoms, \\
             p->enerd,p->fcd,p->lambda,p->graph,p->fr \\
             p->vsite,*(p->mu_tot),p->t,p->field, \\
             p->ed,p->bBornRadii,p->flags);
}
...
// Declare a parameter type and store relevant local data in it
force_params_t p;
...
p.log = fplog;
p.cr = cr;
...
// Pass the function pointer and pointer to the parameter data to MIST
// to be called back when an integrator calls UpdateForces()
MIST_SetForceCallback(do_force_wrapper, &p)
```

By this point MIST has all the data required to carry out a single MD step. We note that, if for any reason the data pointers passed to MIST become out-of-date for any reason such as reallocation, or because the force parameters have changed, the respective `MIST_Set_*` functions must be called again as required.

If an error occurs at any stage (for example if the input file contained unrecognised keywords), the API returns an error code which the caller can check and print an error message, or exit. If the API call was successful, MIST will return `MIST_OK` (`0`).

In order to make using a MIST integrator as intuitive as possible for the user, as much as possible of the unmodified code in the core MD time stepping loop is employed. In particular, if trajectory output and computation of thermodynamic

variables such as temperature are done at the start of the step before a 'native' integrator updates the system state, MIST does the same. If they are done at the end, MIST does likewise. However, in place of the native update code a call is inserted to `MIST_Step()`. This hands over control to MIST to make whatever sequence of updates are implemented in the selected `Integrator`, including calls to the force callback routine as required during the step. When `MIST_Step()` returns, the system state has been advanced by a single time step `dt` and any normal end-of-step actions which are required such as trajectory output and incrementing step counters finally takes place. In the MIST framework, the library is responsible only for the integration step itself. This allows for a separation of concerns between configuring the integrator (via `mist.params`) and run control parameters e.g. number of steps, time step, output frequency and format, which are configured as usual for the host MD code.

Once the simulation has finished, the MIST library can be finalised by a call to `MIST_Cleanup()`, which simply deallocates any memory which has been allocated to allow a clean shutdown of the MD code.

3.4.2 Units

One of the objectives of MIST was to make development of new integrators easy, and to enable a single implementation to be reused with multiple MD codes. In addition to the abstractions discussed already, care is needed to take account of the different units systems in use across different MD codes. To avoid having to include code-specific scaling factors in the `Step()` function of individual integrators, integration takes place using the same units system as the host code, and where parameters are read from the `mist.params` file, the parameter is defined to be in a particular unit and is rescaled into the internal units system using a series of convenience functions. For example, in NAMD-Lite, energies are in kcal/mol, time is in femtoseconds, distances are in Angstroms. To obtain a consistent units system, the internal mass unit is 0.0004184 amu i.e. a hydrogen atom has a 'mass' of 2390.057... internal units. Conversely, Amber uses a units system where masses are in amu, distance is in Angstroms, energies are in kcal/mol, and time is in units of 1/20.455 ps! The relevant conversion factors are defined in the code-specific adaptor classes. LAMMPS is sufficiently general that the user can choose from a range of 8 different unit systems - to support this I added an API function `MIST_SetUnitSystem()` to allow the unit

conversion factors to be provided at runtime, depending on the choice of the user in the LAMMPS input file.

The `System` class defines functions which return standardised lengths (1 Angstrom), masses (1 amu), times (1 picosecond), pressures (1 Atmosphere) and Boltzmann's constant in the internal units system of the host code - with the values provided by the code adaptor classes. These can then by used by `Integrator`s to rescale input parameters into the correct units system. For example, in the Langevin dynamics integrator (Section 3.3.2) the constants $e^{-\gamma\delta t}$ and $\sqrt{k_B T(1-e^{-2\gamma\delta t})}$ are required. The friction parameter γ is converted from the specified units of ps^{-1} into internal time units by:

```
double gdt = friction / system->Picosecond() * dt;
```

Similarly, the Boltzmann factor $k_B T$ is converted into internal energy units (T in Kelvin) and the two required constants are computed by:

```
double kbt = temp * system->Boltzmann();
...
double c1 = exp(-gdt);
double c3 = sqrt(kbt * (1 - c1 * c1));
```

3.4.3 MD code support

MIST currently contains support for five different MD codes. As mentioned in Section 3.2, to add support for each MD code requires inserting MIST API calls into the host code, and implementing an adaptor `System` subclass. The only other code-specific section within the MIST library is the initialisation of the relevant `System` subclass within `MIST_Init()`. Within the general framework laid out in Section 3.4.1, the exact details vary depending on the design and implementation details of the host code. As a result, it is most instructive to learn by example, so the key features of each MD code's MIST integration are described in the following section as a reference to future developers.

NAMD-Lite

NAMD-Lite [93] is the simplest MD code, with no parallelisation and thus provides a straightforward exemplar of how to link MIST with an MD code. It is a modular C code which already provides support for a number of different timestepping methods which can be selected by the user in the NAMD-Lite input file. An input keyword `mist on` is added and if it is present in the input file, a set of initialisation (`step_init_mist()`), compute (`step_compute_mist()`) and cleanup (`step_done_mist()`) functions are chosen in place of the existing NAMD-Lite in-built functions.

These are implemented in a separate file `src/step/mist.c`, which contains the majority of MIST-specific code. The initialisation and cleanup routines are called once at the beginning and end of the MD run, and call `MIST_Init()` and `MIST_Cleanup() respectively`. The compute function is called once and runs a specified number of time steps. Within it, NAMD-Lite first calls the `MIST_Set_*()` API functions to pass in the number of atoms, pointers to the position, velocity and force arrays, a pointer to the variable where the potential energy is stored. NAMD-Lite has a single function which calculates the forces on particles `step_force(Step *s)`, which takes a `Step` struct containing all the parameters required. A simple wrapper function is created which takes an opaque `void *` as required by the MIST API, casts it to a `Step *` and calls `step_force()`. This wrapper function and a pointer to the `Step` object are passed in to the `MIST_SetForceCallback()` API function. The NAMD-Lite topology data is obtained and iterated through, calling `MIST_SetNumBonds()` and `MIST_SetBond()` once for each bond in the topology. Once the topology data has been passed in to MIST, `MIST_GetNDoF()` is called since if the MIST input file specified the use of bond constraints, this reduces the number of degrees of freedom, and the starting velocities must be rescaled accordingly.

Once all of this initialisation is complete the main computation is simply a loop of calls to `MIST_Step()` with a call to the NAMD-Lite output function `step_results()` every `resultsFreq` steps, exactly the same as for any other `step` implementation.

All of the MIST API calls are wrapped in with an error-checking function `MIST_chkerr()` which checks the return code from the API call, and if it is not `MIST_OK`, prints an error message and exits.

There are various other small modifications required throughout the rest of the NAMD-Lite source code. These can easily be found as they are all marked by a `// MIST patch ... // End of MIST patch` pair of comments. These are mainly concerned with adding additional 'plumbing' to allow access to the various pieces of data such as the individual components of the force-field that MIST needs access to. The build configuration files are also modified to ensure the MIST header file is included during the compilation phase, and the MIST library is linked into the final product executable.

One somewhat complex aspect of the implementation is support for the `MIST_FEATURE_POS_VEL_OFFSET_PLUS_HALF_DT` feature flag. At the start of the run, if this feature flag is returned by `MIST_GetFeatures()` an additional velocity half-step is made, and the `ave_ekin` variable is set to `TRUE`. If this is the case, the velocities are stored at the end of each step, and in the routine which calculates the kinetic energy the average of the preceding $(t - dt/2)$ and current $(t + dt/2)$ kinetic energies is used to approximate the kinetic energy at the end of the step.

NAMD-Lite also makes an inconsistent choice of units system - with masses in AMU, time in fs and energies in kcal/mol/K. In the native integrators in NAMD-Lite this is requires the computed forces to multiplied by a constant `MD_FORCE_CONST` ≈ 2390.06. Since MIST integrators are designed to be agnostic of the host units system, the stored masses are divided by `MD_FORCE_CONST` before passing them to `MIST_SetMasses()`, achieving the same net effect.

The `NAMDLiteSystem` subclass within the MIST library is quite straightforward. Units conversion factors are set in the constructor. The positions, velocities and forces and masses are stored in double-precision arrays, and the other data structures such as the lattice vectors and the force components are simply cast to their respective data types and unpacked. NAMD-Lite has no support for changing the cell dimensions at runtime, so `SetCell()` simply prints a warning message and does nothing.

GROMACS

MIST currently includes a set of source code patches which implement support for MIST in GROMACS [4] version 5.0.2. While this version dates from 2014, it contains many of the major features of recent versions of GROMACS such as parallelisation with MPI, OpenMP and GPU acceleration. The GROMACS

code-base was largely refactored and converted to C++ in the 2016 release and while maintaining MIST support would certainly be possible, development of the library and adding support for different MD codes was prioritised over keeping up with the latest versions as they were released.

GROMACS presents quite a different set of challenges to NAMD-Lite. With over 2,200,000 lines of code, compared to 86,000 it is clearly much more complex and feature-rich. Rather than NAMD-Lite which had a fairly modular structure, GROMACS is monolithic with a main MD loop comprising over 1200 lines with complex logic covering Velocity Verlet, Velocity Verlet with Kinetic Energy averaging, stochastic (Langevin) Dynamics, plus myriad temperature and pressure control schemes, parallelisation concerns such as domain decomposition and load balancing, as well as diagnostic and statistical output! Since MIST is designed to tke control of the time stepping process, the simplest solution for integration with GROMACS is to simply create a top-level branch within the timestepping loop to cover the case where MIST is selected containing only the functionality required by (and compatible with) MIST, and the standard MD loop remains unmodified. Thus the overall structure is as shown:

Listing 3.13: MIST integration in GROMACS main MD loop in `md.c`

```
double do_md(...)
{
  // Standard GROMACS initialisation code
  ...

  // MIST Initialisation
#ifdef __MIST_WITH_MPI
  MIST_chkerr(MIST_Init_MPI(cr->mpi_comm_mygroup),__FILE__,__LINE__);
#else
  MIST_chkerr(MIST_Init(),__FILE__,__LINE__);
#endif
  ...

  // Check for GROMACS input parameters known to be incompatible with MIST
  ...

  // Standard GROMACS MD loop setup
  ...

  // Main MD loop
  while (!bLastStep || (bRerunMD && bNotLastFrame))
  {
    // Do a minimalist MD step with MIST
    if (ir->eI == eiMIST)
    {
      ...
      // Call into MIST to execute the step
      MIST_chkerr(MIST_Step(ir->delta_t),__FILE__,__LINE__);
      ...
    }
    else
    // Do the normal (non-MIST) MD step
    {
      // Unmodified GROMACS MD step code
      ...

    }
  }
  /* End of main MD loop */

  // Clean up MIST
  if (ir->eI == eiMIST)
  {
    MIST_chkerr(MIST_Cleanup(),__FILE__,__LINE__);
  }
}
```

The overall ordering of MIST operations is very similar to NAMD-Lite, but the key differences are described below. Firstly, the pre-compiler macro `__MIST_WITH_MPI` which is set during the build process is used to allow the inclusion of both MPI-parallel and serial code within the same code base. The input parameter check `ir->eI == eiMIST` (corresponding to `integrator`

= mist in the GROMACS .mdp input file) is used to branch between MIST and non-MIST code paths. A number of input parameters which are not allowed, or don't make sense to be used if MIST is selected are checked up-front, including solid 'walls' at the faces of the simulation box, temperature and pressure control within GROMACS (since this is done by MIST), core-shell ion models (since MIST assumes only point particles), and water-ion swapping amongst others. The associated code for these is removed from the MIST-branch of the main loop, simplifying the implementation significantly.

One other major difference to NAMD-Lite is related to MPI parallelisation. Every nstlist time steps (or when particles are found to have moved outwith the neighbour list buffer region), GROMACS rebuilds the neighbour list which is used by the force routines to determine which particles are within range and should be considered when calculating the non-bonded interactions. At the same time, GROMACS also performs a repartitioning in order to maintain good load balance. After the repartitioning, the number of particles assigned to each process may have changed, and so MIST_SetNumParticles() must be called again. In addition MIST_SetPositions() and the other setters must also be called because the arrays storing the particle data may be reallocated, invalidating the pointers that were passed to MIST during initialisation.

The only other major change in the GROMACS code is that the GROMACS force routines do not contain separate arrays for storing the forces due to different terms in the force-field. While the energies are accumulated separately, if MIST is using an integrator which requires access to each component of the force-field (i.e. the flag MIST_FEATURE_FORCE_COMPONENTS is returned by MIST_GetFeatures(), arrays are allocated to store each of different force-field components required (bond, angle, proper and improper dihedrals, and non-bonded), store pointers to them in a struct, and that struct is passed into the normal GROMACS force computation routines, which are modified to write each part of the force into the relevant array.

While the structure of the MD loop is very complex, GROMACS does successfully abstract the use of GPUs to accelerate the force calculation - and thus MIST is completely agnostic to this. If GROMACS is compiled with CUDA support for Nvidia GPUs then they will be used by the force routines without any code changes required in the MIST MD loop.

In the GromacsSystem adaptor class within MIST, there are few surprises.

GROMACS stores all of its state variables in single precision, only making use of double precision for intermediate quantities during the force calculations. Each of the MIST System accessors must cast to or from double precision as needed. Since only the inverse mass $1/m_i$ appears in the update step $\ddot{r}_i = F_i/m_i$, GROMACS does not store the particle masses directly but rather the inverse mass. So the array passed in to MIST by the `MIST_SetMasses()` API call must be inverted and stored if the mass is required by the MIST integrator. Finally, since GROMACS supports a variable simulation cell the `GetCell()` and `SetCell()` functions are implemented, following the simple 3-step process described in Section 3.3.3.

Amber

MIST support for Amber [48] is based on Amber 14 and is included in the optimised, parallel MD program `pmemd`. Similarly to GROMACS, while this version is now several years out of date, it serves to illustrate the use of the MIST Fortran 90 API and also some different approaches to parallelisation and GPU acceleration compared with GROMACS.

A broadly similar approach to GROMACS is taken, with an additional input parameter `imist .eq. 1` being used to select MIST as opposed to the native Leapfrog integrator. Again, a number of features of Amber are explicitly disallowed from being used in combination with MIST, including test-particle insertion, frozen 'belly' restraints, centre-of-mass motion removal and various temperature and pressure control algorithms. Like GROMACS, this reduces the amount of code required to be included in a MIST version of the main MD loop by more than half.

There are two main complexities that needed to be overcome to have MIST working correctly with Amber. Firstly, the load balancing algorithm is done within the routine `pme_force()` which is also used to compute the forces on each atom. Since MIST requires that particles are not migrated during the scope of a single MD step, the force routines were modified to take an additional, optional parameter which turns off the load balancing functionality when called from within the MIST force callback. However, load balancing should still be performed in order to maintain good parallel performance, so the ordering of operations is modified to:

Listing 3.14: MIST integration in Amber main MD loop in `runmd.F90`

```
double do_md(...)
{
! Main MD loop

if (imist .eq. 1) then
  do

  #ifdef MPI
    ! If a redistribution is needed, then do it here, also getting updated
    ! forces for the newly localised atoms:
    if (using_pme_potential .and. new_list .and. &
      (atm_redist_needed .or. fft_redist_needed) ) then
        ...
        ! We don't actually want new forces or energies
        ! Just want to do the redistribution, and associated book-keeping
        call pme_force(atm_cnt, crd, frc, gbl_img_atm_map, gbl_atm_img_map, &
                       my_atm_lst, new_list, .false., .false., &
                       pme_pot_ene, virial, ekcmt, pme_err_est)
        ...
        ! Update MIST with the new number of local atoms
        call MIST_chkerr(MIST_F_SetNumParticles(my_atm_cnt), __LINE__, __FILE__)
    endif
  #endif
    ...
    ! Do the step within mist.  Force callback calls pme_force with
    ! allow_load_balancing = .false.
    call MIST_chkerr(MIST_F_Step(dtx), __LINE__, __FILE__)
    ...
    ! Book-keeping and output code
    ...
  end do ! Major cycle back to new step unless we have reached our limit

  call MIST_chkerr(MIST_F_Cleanup(), __LINE__, __FILE__)

else ! No MIST
  do
    ! Unmodified Amber MD loop
    ...
  end do
}
```

Secondly, Amber's approach to GPU implementation (described in Section 2.1.4) achieves high performance by doing not only the force calculation but also the particle position and velocity updates all on the GPU. This presents a problem for MIST, where everything except the force calculation is done inside the MIST library (and hence on the CPU). Whereas Amber only downloads data from the GPU when required for output, MIST requires data transfer at every step - comparable to other codes like GROMACS. As a result, in the force callback routine the latest particle positions from the arrays accessed by MIST must be uploaded to the GPU, `pme_force()` called to calculate the forces, then the

computed forces from the GPU must be downloaded so they can be read by MIST when the callback returns. As shown in Section 4.3.3, while this ensures the calculation is correct it does result in a significant performance impact.

Forcite

Forcite [30] is a proprietary MD code developed by Dassault Systems BIOVIA (formerly Accelerys) and sold as the classical MD engine within their Materials Studio product. Funded by an EPSRC Impact Acceleration Account project "A Flexible Software Interface for Molecular Modelling", I developed an interface between Forcite and MIST during a week-long visit to BIOVIA's development team in Cambridge in January 2017. The project successfully demonstrated that Forcite could make use of MIST integrators - in particular the Langevin Dynamics integrator with Symmetric Newton constraints solvers. Unfortunately, due to the confidentiality agreement in place for the project the resulting source code and test results are retained by BIOVIA. Nevertheless, this shows that (a) some of the algorithms implemented in MIST are of practical interest to the industrial as well as academic molecular simulation community and (b) that the MIST API is flexible enough that it can be used to add support into a previously unseen code within a very short space of time.

LAMMPS

LAMMPS [174] has a very different structure to either GROMACS or Amber as it takes a modular approach which makes it easy to extend by introducing new code via optional user packages. Unlike the other codes, instead of source-code patches which modify the core MD loop, I implemented MIST support in a separate package `USER-MIST`, which introduces a MIST `run_style` and a `fix`. As LAMMPS development is done openly on github.com I made a fork of the LAMMPS code-base and distribute the modified version at `https://www.github.com/ibethune/lammps/tree/mist`. Eventually, this is expected to be merged into the main LAMMPS distribution via a Pull Request.

Compared with the default LAMMPS run, which allows the user to select an arbitrary set of `fix`es which can be mixed and matched to perform the required sequence of updates (e.g. velocity verlet plus a nose-hoover thermostat), the MIST run class initialises the MIST `fix` (and no others), initialises the

MIST library and simply carries out the specified number of timesteps, calling `MIST_Step()`. Similarly to Amber, load balancing (if required) is done at the start of the step. Unlike most LAMMPS `fix`es, the MIST `fix` does not do any updates to the state of the system, but is simply used to ensure that the particle forces are packed into the arrays LAMMPS use for communication during load balancing. LAMMPS also provides several `compute` classes which can be used (if needed by the MIST integrator) to calculate quantities such as the total potential energy or the pressure. By taking advantage of as much as possible of the normal LAMMPS infrastructure, it is possible to provide a fully-functional, MPI-parallel MIST implementation in less than 1000 lines of code.

One other unique feature of the implementation is because LAMMPS can run using one of 8 different units systems, `MIST_SetUnitSystem()` is called immediately after initialising MIST, as opposed to setting the units scaling factors in the `System` adaptor class as is done for other codes.

3.5 Building and running MIST

In order to use MIST with a particular MD code requires compiling the code with MIST API calls inserted into the source code and linking the MIST library containing the corresponding `System` adaptor class. For codes which MIST provides adaptors for, source code patches are included with MIST which can be applied during the build process (with the exception of LAMMPS where I distribute a version containing MIST calls). To a user, the process is very simple:

- Configure MIST for use with a particular MD code, providing the location of the source code as an argument to the MIST `configure` script e.g. `--with-amber=/path/to/amber/src`. The only other configure option is whether to enable MPI support (`--enable-mpi`). If configuration was successful, the script provides step-by-step instructions to complete the build for that specific MD code. In general, the steps proceed as follows:
- Generate and apply the source code patch. For the versions that MIST supports (NAMD-Lite 2.0.3, GROMACS 5.0.2, Amber 14), the code patches will apply seamlessly. It may be possible to apply the patches to other similar versions.

- Build the MIST library. Using a Makefile, MIST is built with support for the selected MD code and either serial or MPI support compiled in.
- Build the host code. The host code is built (including the inserted MIST API calls) and linked with the MIST library. Appropriate modifications to the build options are automatically made by the patching process, so no manual configuration such as library search paths or other linker flags is required from the user.

Once the MD code is built, it can be used entirely as normal if desired. To use a MIST integrator instead of the native ones provided by the host code requires adding a single parameter in the input file:

- NAMD-Lite: add `mist on` to the `.config` file.
- GROMACS: set `integrator = mist` in the `.mdp` file. Note that this change should be run through the GROMACS preprocessor `grompp` as usual, in order to have any effect.
- Amber: set `imist = 1` in the `&cntrl` namelist in the input file read by `pmemd`.
- LAMMPS: set `run_style mist` in the LAMMPS main input file.

If MIST is enabled, execution will continue as described in Figure 3.2 and timestepping will be carried out according to the integrator and options specified in the `mist.params` file which is read by MIST. A separate file is used so that integrator settings are code-independent, whereas run control settings such as the number of time steps, when to write trajectory output etc. are managed by the usual input file(s) of the host code. Any trajectory or diagnostic output options specified in the host code's input file will still produce output in the usual format since MIST only modifies the dynamics.

See Section 3.3 for details of each integrator and the parameters that it takes, but a minimal MIST input file to use velocity Verlet integration in the NVE ensemble would be:

```
integrator verlet
```

A slightly more complex example, to run Langevin NVT dynamics at 300K would be:

```
integrator langevin
langtemp 300 # Target temperature, in K
langfriction 1.0 # Friction parameter gamma, in ps^(-1)
```

Chapter 4

Evaluation

This chapter provides evidence that MIST works correctly, assesses the performance of the library, and illustrates its use by exploring the free energy landscape of Alanine-12 using the Simulated Tempering algorithm. Using these results and my experience working with collaborators using and developing algorithm in MIST, I review the library against the design goals set out in Section 3.1.

4.1 Automated Tests

The importance of testing software as it is developed, rather than after the fact is well-known best practice in software engineering. By catching bugs early it makes identifying the cause and fixing it easier and faster. The practice of *Continuous Integration* involves defining a pipeline of automated actions (including tests) which are run after every change to the code committed to a version control repository. MIST makes use of Bitbucket Pipelines, the native CI solution provided by the source code hosting service. A pipeline of build and test actions is defined and is run in a controlled environment (using Docker containers) after every commit. Any failures are reported immediately to the developer and the commit is marked as failed.

The pipeline consists of three stages, run sequentially. The sub-steps at each stage are run in parallel:

1. Build a MIST distribution package, and check that the entire code base

is fully documented with Doxygen. This ensures that the code is always kept in a state that it can be distributed. The automated documentation check is particularly useful in maintaining code quality as documentation is easy to omit, but this ensures that it is kept up-to-date as the library is developed. If these checks pass, then...

2. Build MIST and link it to each of the supported codes, including serial and parallel compilation. This proved invaluable during the development of MIST support for each code, as it ensures that any changes made to the library do not break compilation of any of the other codes. Similarly, as MPI parallelisation was added it ensures that serial compilation was not affected. Once all the builds are successful...

3. Run a series of short MD runs and check the results. A simple check of the total energy at the end of the run is used. This test gives some confidence that any changes made to the library do not affect the numerical results i.e. that the behaviour of the integrators remains correct. At present, two different inputs - a system of 32,000 Lennard-Jones particles and one of a small peptide solvated in water (2004 atoms) - are used, and both run with and without MIST, in serial and parallel, using LAMMPS.

A more extensive set of test runs and sample outputs are included in the `mist/tests` directory. A set of shell scripts are provided which build the library and MD codes, run the tests and report any results which deviate from the expected reference results. These were developed early on in the project before a controlled CI environment using Docker was available and so are hard-coded to work on a specific development machine.

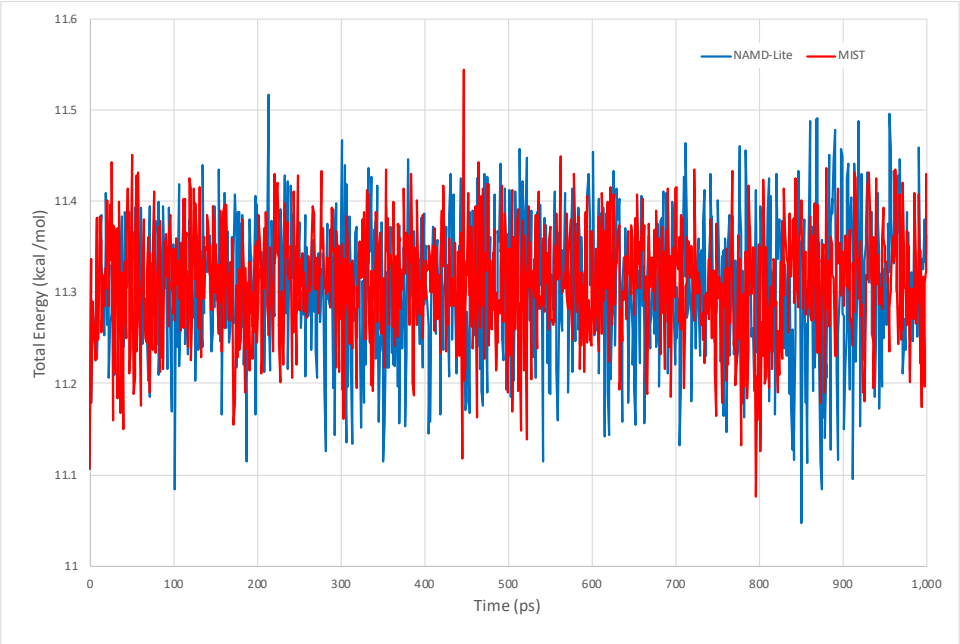
During the 3 years of development where CI has been in place, 25.6% of commits have resulted in a test failure, which was immediately caught and fixed - clearly automated testing does help to improve code quality!

The current CI pipeline could be extended to include numerical validation of results for all the supported codes, to give greater confidence that library changes in support of one code do not affect the others. However, this has not yet been implemented.

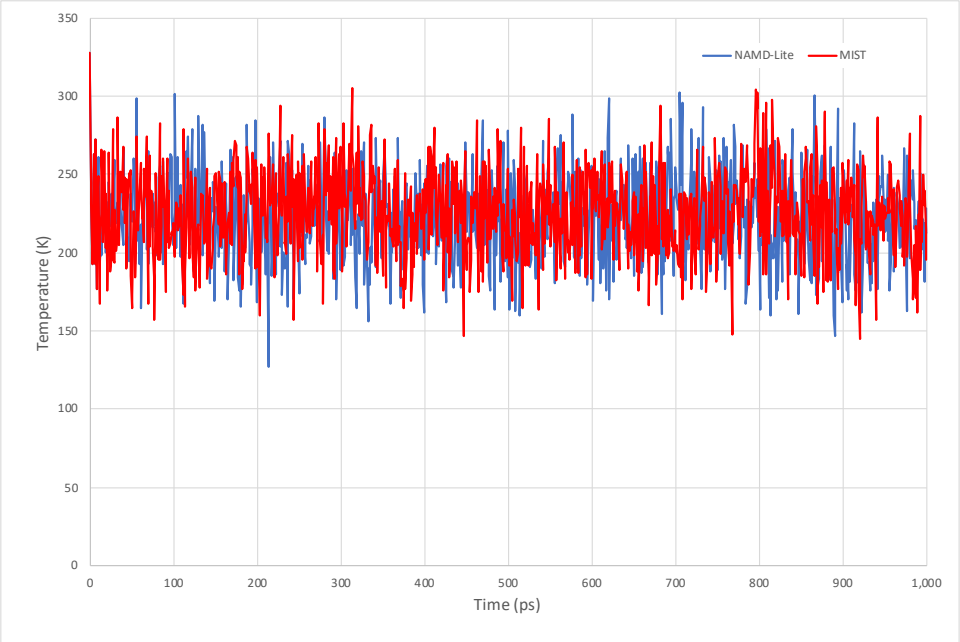
4.2 Validation

Two main approaches have been taken to show that MIST behaves correctly - firstly like-for-like comparison can be made between native integrators in each MD code and their counterparts implemented within MIST. For these tests the behaviour of the system being modelled is not particularly important so it is not necessary that 'good' quality parameters are chosen. The output of host MD code is taken as a baseline, and the results achieved using MIST are compared against it. Even where the algorithm implemented in MIST is analytically identical to the native integrator small numerical differences such as the use of double or single-precision, or even the ordering of arithmetic operations within the individual particle updates will cause the results of the calculation to differ. As a result the correctness of the MIST implementation can be assessed by comparing the long-term trends in computed quantities such as the temperature, and by checking that expected quantities such as the total energy (for a simulation in the microcanonical ensemble) are conserved. This approach can also be used to check that parallelisation is implemented correctly. Secondly, for integrators where there are no exact counterparts implemented in any of the supported codes, expected properties of the algorithms can be checked, for example how well they conserved energy, or (for a simulation in the canonical ensemble)whether temperature converges to the expected value. An illustrative selection of both of these types of test result are shown in the following section, some for each of the supported MD codes. The input files used are distributed with MIST, in the `examples` subdirectory.

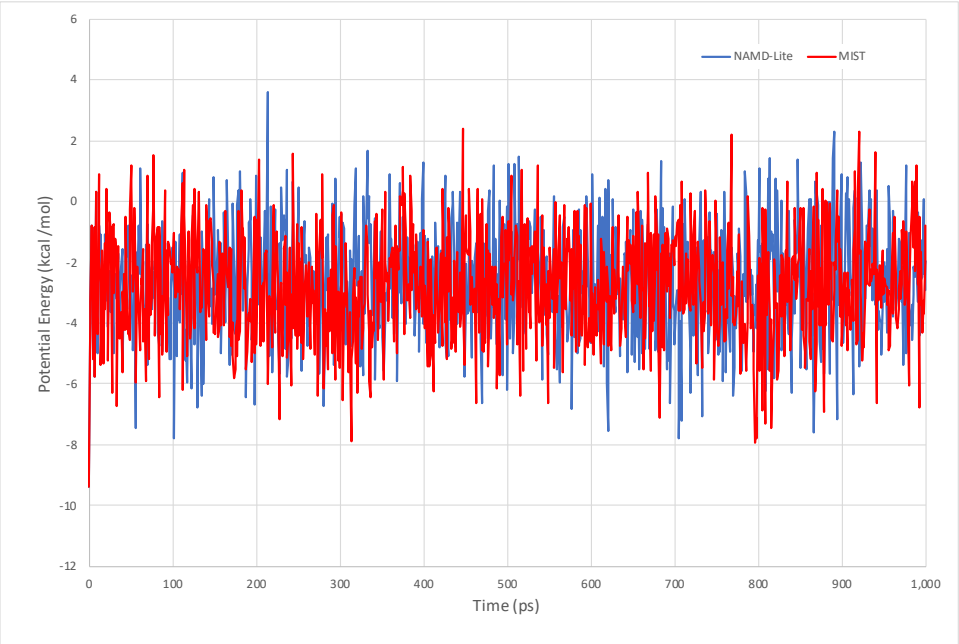
To validate the NAMD-Lite implementation, a small molecule - adenosine diphosphate - isolated in vacuum with no periodic boundary conditions and the CHARMM22 All-Hydrogen forcefield parameters [3] are used, with no constraints of any kind. The atoms are initialised with velocities at 300K and then run for one million 1 fs time steps in the NVE ensemble for a total of 1 ns of dynamics. The input files can be found in `examples/namd-lite/adp` in the MIST distribution. The default `verlet` integrator from NAMD-Lite is compared against the `VerletIntegrator` in MIST in Figure 4.1. For such a small system (only 22 atoms) large fluctuations are expected in all computed quantities. However, it's clear that the total energy for both integrators is around 11.3 kcal/mol. Both the kinetic energy / temperature and the potential energy are also in good agreement, indicating that MIST is integrating the equations of motion correctly.



(a) *Total Energy Conservation*



(b) *Temperature*

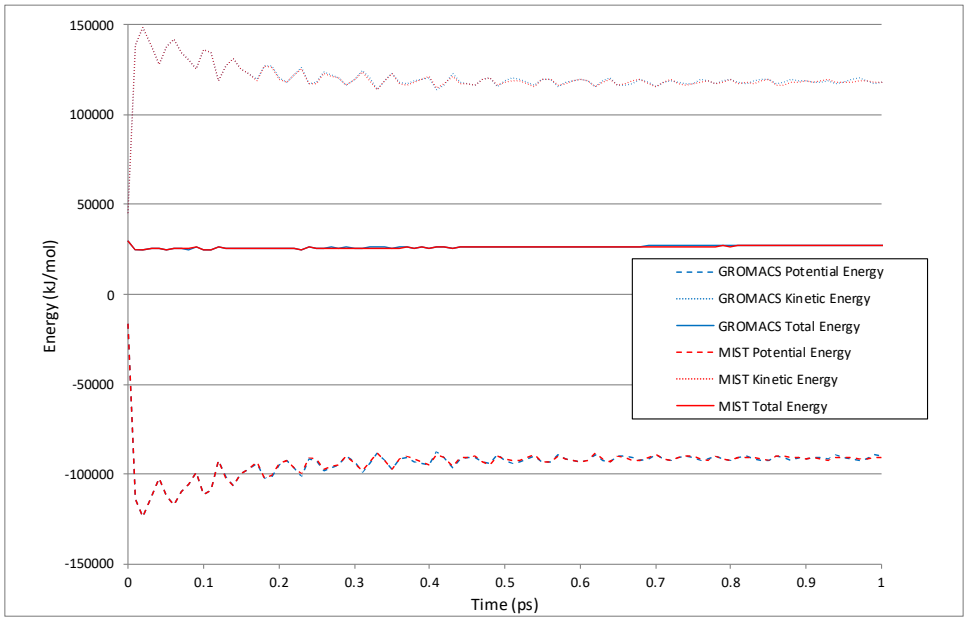


(c) *Potential Energy*

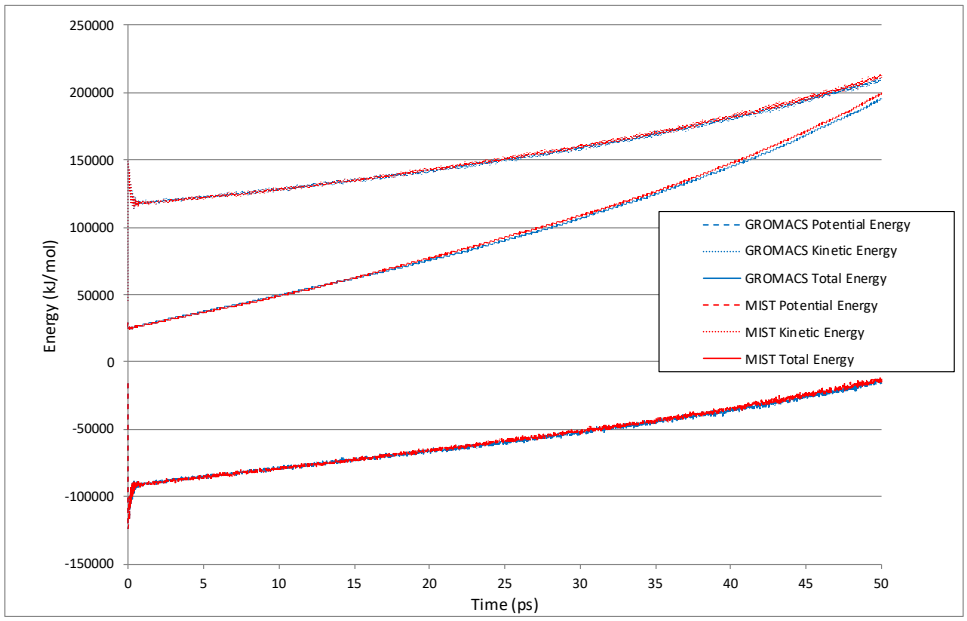
Figure 4.1 *Comparison of NAMD-Lite and MIST Verlet integrators for ADP* in vacuo.

As a production-quality MD package, GROMACS achieves higher performance and is able to model significantly larger systems where energy fluctuations are expected to be much smaller and therefore it is easier to compare MIST with the native integrator. The `examples/gromacs/water` directory contains input files for a 12207-atom system of water molecules in a 5nm cubic periodic box. The GROMACS implementation of the CHARMM27 forcefield [31] is used with fully flexible TIP3P water and a cutoff of 1.2nm for non-bonded electrostatic and van der Waals interactions. Production calculations would typically use a more reliable method such as Ewald summation or similar for the long-range non-bonded forces rather than a bare cutoff [172] as these are known to produce better simulation results, and may contribute to the drift in total energy observed in Figure 4.2b. In any case, excellent agreement between GROMACS' native Verlet scheme and MIST's `Verlet` can clearly be seen for both the kinetic and potential energy, and thus the total energy. In particular, they are virtually indistinguishable for the first 1000 steps or more (Figure 4.2a) before numerical differences cause a small divergence over the remainder of the 50ps of calculation.

A similar system is set up in Amber, using the same initial coordinates but a different force-field. In this case the Amber ff03 [60] forcefield was used, with a relatively high cut-off of 24Å(full parameters used can be found in



(a) *Initial 1ps (Zoomed in)*



(b) *Full 50ps calculation*

Figure 4.2 *Comparison of GROMACS and MIST Verlet integrators for liquid water test case.*

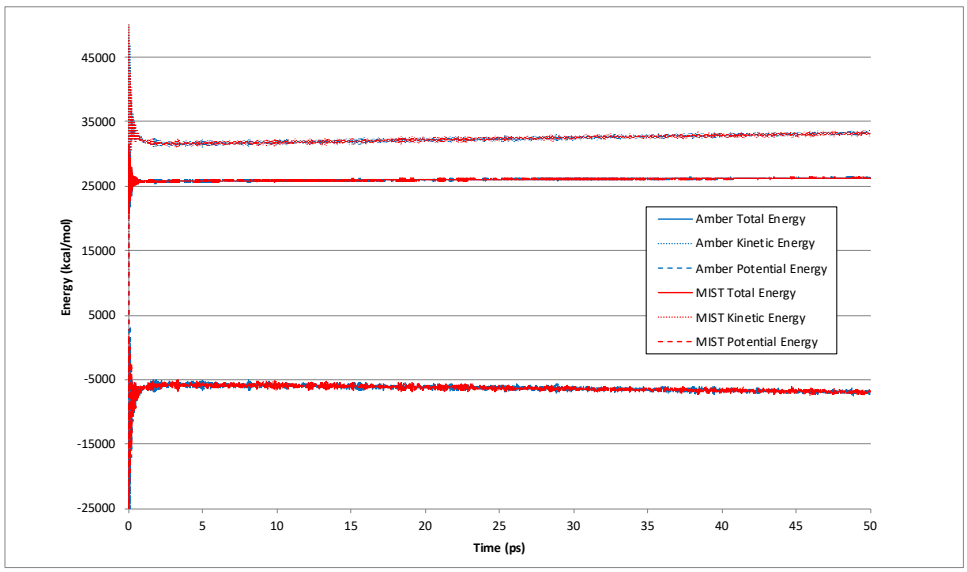
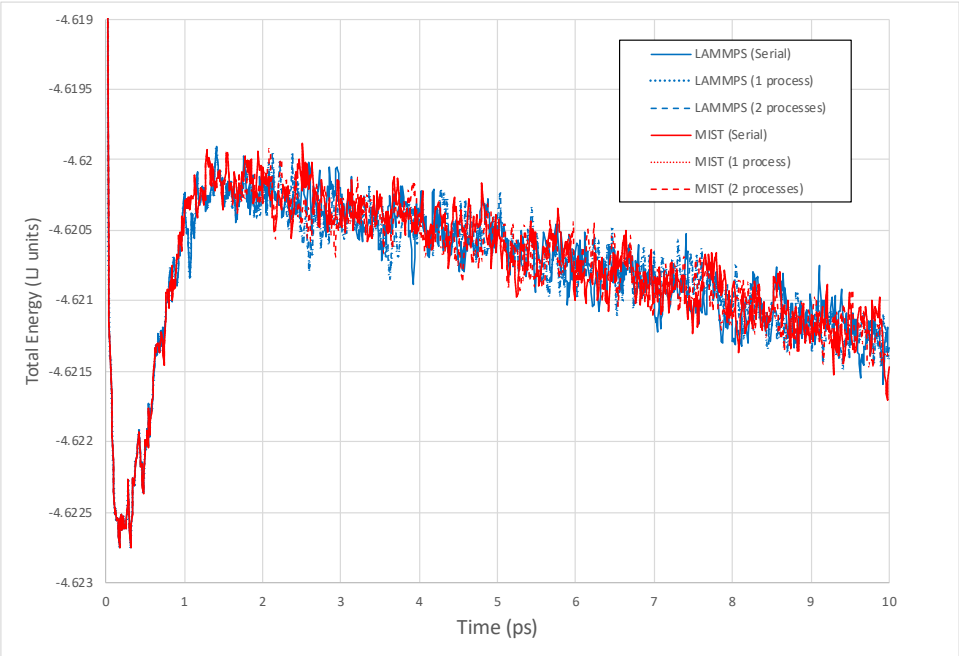


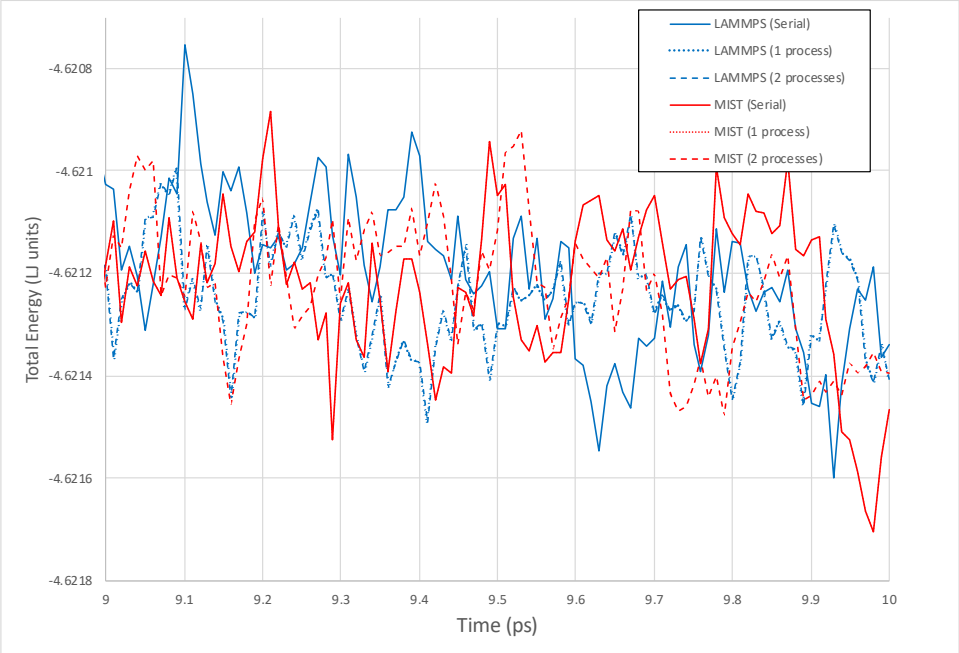
Figure 4.3 *Comparison of Amber and MIST Leapfrog integrators for liquid water test case, running on 2 MPI processes.*

`examples/amber/water`). The native integrator in Amber is a Verlet leapfrog scheme, so the `LeapfrogIntegrator` in MIST is used for comparison, and in addition both native and MIST calculations were run using 2 MPI processes to validate the parallel implementation of MIST. The data shown in Figure 4.3 shows excellent agreement between Amber and MIST integrators for both kinetic and potential energy, and much better total energy conservation than the parameters selected for the test in GROMACS (although in both cases MIST shows the same behaviour as the native integrator).

For validation of LAMMPS, one of the standard LAMMPS benchmark cases is adapted (see input files in `examples/lammps/lj`), which consists of 32,000 Lennard-Jones particles arranged in a periodic FCC lattice. 5000 steps of NVE dynamics were run using the LAMMPS `fix nve` and MIST's `VerletIntegrator`. LAMMPS and MIST were compiled both with and without MPI support and MPI runs were done on both 1 and 2 processes to examine the numerical effects of parallelisation. Figure 4.4a shows that all six runs are in very close agreement on the total energy and are all considered to be correct. Interestingly, closer examination of the data at the near the end of the run (Figure 4.4b) shows that while there are small numerical differences between the LAMMPS serial and parallel runs, LAMMPS in parallel gives identical results to within machine precision irrespective of the number of processes. In contrast, MIST gives identical results when run in serial, or with MPI on a single process, but slightly different results when run on 2 processes.



(a) *Full 10ps calculation*



(b) *Final 1ps (Zoomed in)*

Figure 4.4 *Comparison of LAMMPS and MIST Verlet integrators for the Lennard-Jones fluid test case.*

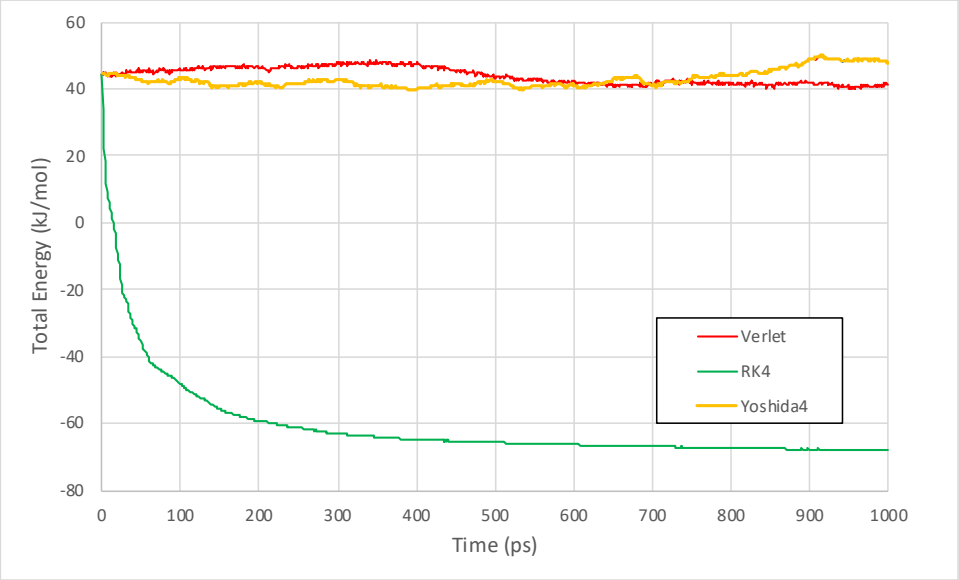


Figure 4.5 *Total Energy conservation of different NVE integrators for the alanine dipeptide test case.*

As discussed in Section 2.1, Verlet integration is the preferred scheme for MD due to the favourable trade-off between discretisation error and cost. To demonstrate the behaviours of the `RK4` and `Yoshida4` integrators, a GROMACS simulation of a small (22 atom) molecule of alanine dipeptide in a large periodic vacuum box was set up using the Amber03 force-field with a cut-off of 1.2nm (input files in `examples/gromacs/aladip`). 1ns of dynamics with a time step of 1fs is shown in Figure 4.5 where it is clear that both the Verlet and Yoshida integrators have good long-term energy conservation because of their symplectic structure. Although the Verlet integrator has a larger discretisation error, this is not practically relevant since integrator becomes unstable at any larger step sizes where the Yoshida integrator might otherwise prove advantageous because the system contains fully flexible N-H bonds with an oscillation period of around 10fs. With similar practical energy conservation, Verlet is approximately 3 times faster - 7456 ns/day compared to 2497 ns/day for Yoshida - due to the 3 force evaluations required per time step. While the Runge-Kutta scheme has the same discretisation error as the Yoshida scheme, it is not symplectic and so it expected to have poor energy conservation. However, the extreme loss of energy suggests this is due to an implementation error!

Similarly, the behaviour of the `LangevinIntegrator` is validated using the same liquid water system in GROMACS as described above. A total of 10ps of dynamics were run, varying the target temperature and the friction parameter

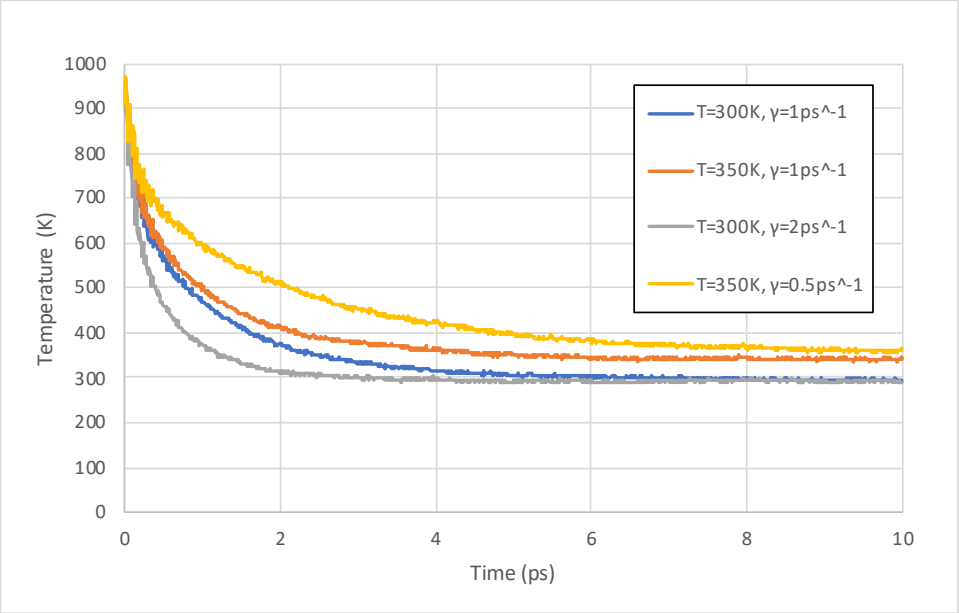


Figure 4.6 *Temperature of the water test case in GROMACS, with various Langevin thermostat parameters.*

γ. The Langevin scheme is expected to relax smoothly to the target temperature as opposed to oscillating around it like the Nosé-Hoover scheme shown in Figure 4.7. Figure 4.6 shows that MIST correctly converges the temperature as reported by GROMACS to the respective targets of 300K or 350K. With the friction set to its default of 1/ps the target temperature is reached after 5-6ps. As expected, increasing or decreasing γ decreases or increases the relaxation time proportionally.

LAMMPS was used to test the MIST `NoseHooverIntegrator` with only the thermostat enabled. An input file which sets up a 5-mer peptide (84 atoms) in a periodic 27.37Å cubic box of 640 water molecules, for a total of 2004 atoms was modified to run for 2,000 2fs steps. The CHARMM22 force-field is used with LAMMPS' Particle-Particle-Particle-Mesh algorithm for long-range electrostatics and all bonds fully flexible. The modified input files can be found in `examples/lammps/npt`. The LAMMPS `fix nve` is used as a baseline, and then `fix nvt` with the target temperature set to 300K and a damping time constant of 200fs. The same temperature was used to configure the MIST integrator and the default `temp_nsteps_relax` of 100 steps. Figure 4.7 shows the temperature computed during each run. The NVE run shows that in absence of any thermostatting, the temperature oscillates around 215K (in the original input file, initial velocities were assigned at 275K, but in this case the same set of velocities in the absence of any bond constraints equates to around 215K due

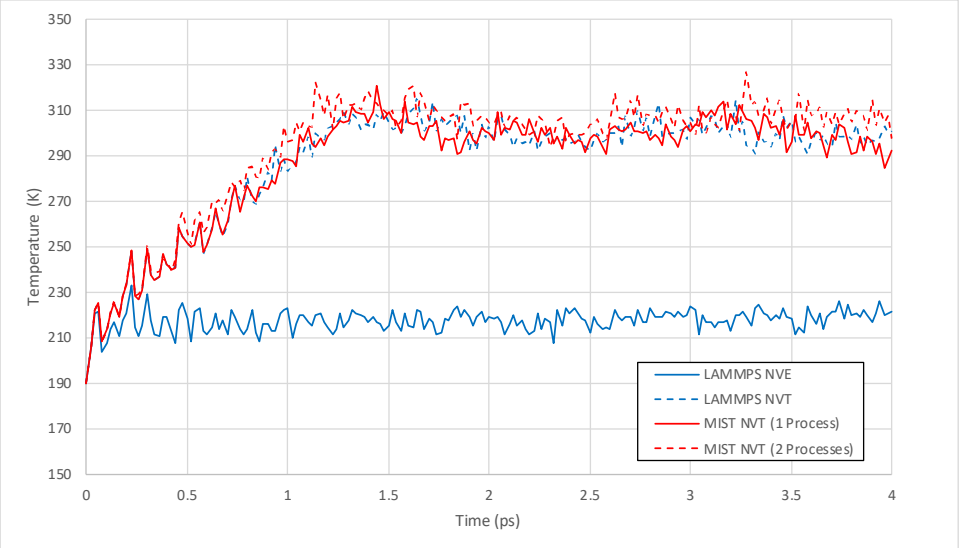


Figure 4.7 *Comparison of LAMMPS and MIST Nose-Hoover NVT integrators for solvated 5-mer peptide test case.*

to the higher number of degrees-of-freedom). LAMMPS and MIST NVE runs have been compared in Figure 4.4 so the MIST run is omitted here for clarity. Both the LAMMPS and MIST thermostats exhibit similar behaviour, reaching the target temperature of 300K after around 1.2ps. In particular, it can be seen that the initial temperature fluctuations match exactly, before subtle differences in the algorithms causes the results to vary, although still correctly maintaining the temperature at 300K.

The same system is also used to validate the behaviour of the barostat. In this case the LAMMPS integrator parameters `fix 1 all npt temp 300.0 300.0 $(100.0*dt) iso 1.0 1.0 $(100.0*dt)` are used, and the same target temperature (300K), pressure (1 Atm) and relaxation time constant (100 steps, or 200 fs) are used for both thermostat and barostat in MIST. Figure 4.8 shows that initially, there are quite large oscillations in the pressure for the first 1 ps or so before these are damped by the barostat and small oscillations around the target remain for the duration of the run. The MIST implementation recreates the same oscillatory frequency as LAMMPS' `fix npt` and both damp magnitude of the pressure oscillations at the same rate.

To test the behaviour of Ackland's box quenching scheme [5], the LAMMPS `examples/meam/in.eam` input file which sets up a system of 32,000 copper atoms in a perfect fcc lattice was used, with Embedded Atom Method interaction parameters taken from [72]. Starting with the lattice parameter expanded by 10%

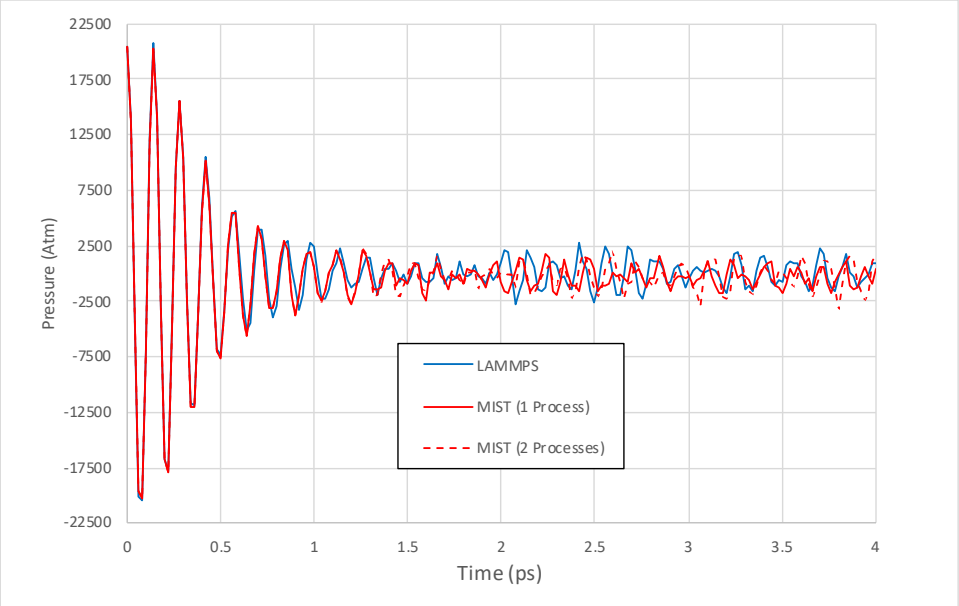


Figure 4.8 *Comparison of LAMMPS and MIST Nose-Hoover NPT integrators for solvated 5-mer peptide test case.*

NPT dynamics are run using a traditional Parrinello-Rahman style barostat (i.e. no thermostat chain attached to the barostat), which sets up a elastic 'ringing' vibration of the cell as shown in Figure 4.9. By comparison, when the same system is run using the MIST NPT integrator with the parameter `box_quench true`, it is clear that as soon as the cell reaches the equilibrium volume during the first contraction it stops contracting and relaxes rapidly. Small fluctuations are due to the fact that during a single time step the cell volume change may overshoot the equilibrium volume slightly.

For the other MIST integrators, Continuous Tempering [81] and ISST [146], validation results can be found in their respective publications. Simulated Tempering is applied to a real system in Chapter 5.

4.3 Performance Testing

Having demonstrated that MIST integrators give correct results, the performance overhead of using MIST compared with the native integrators in each of the supported host codes can be investigated. As MIST introduces additional layers of abstraction, these are expected to come at a computational cost, but the design choices laid out in Section 3.1 are intended to keep this to a minimum.

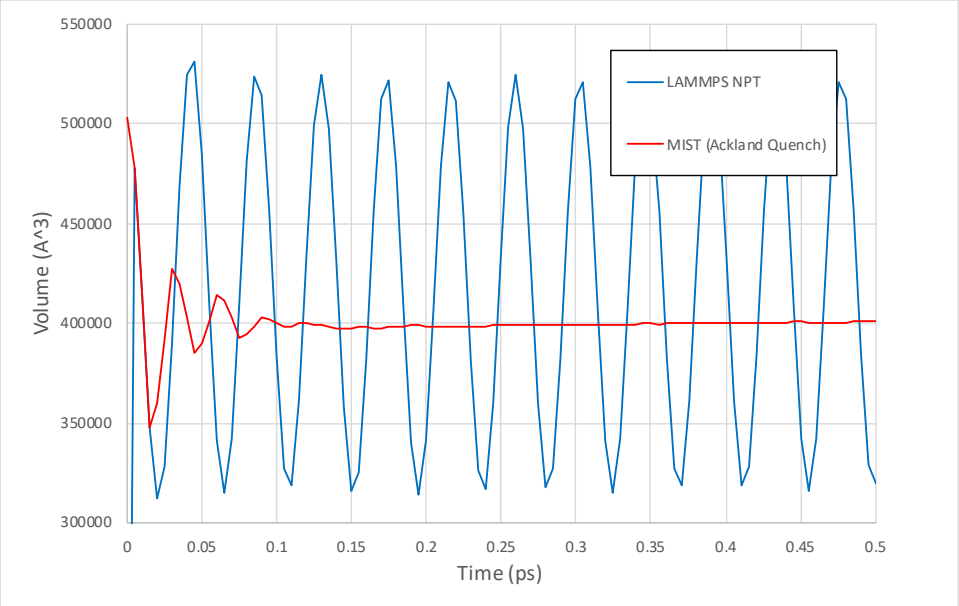


Figure 4.9 *Comparison of LAMMPS NPT and MIST NPT with Ackland's box quenching scheme for copper crystal test case.*

Unless otherwise noted, CPU tests have been performed on ARCHER, a Cray XC30 with two Intel Xeon 12-core E5-2697v2 'Ivy Bridge' processors per node. Some of the MPI-parallel tests were carried out on Cirrus, a SGI/HPE 8600 cluster with two Intel Xeon 18-core E5-2695 'Broadwell' processors per node. GPU tests have been performed on a Linux system with two Intel Xeon 8-core E5-2650v2 'Ivy Bridge' processors and eight NVIDIA Tesla K40m GPUs. GROMACS uses one GPU per MPI rank even if that process makes use of multiple CPU threads.

All bar graphs show the average over 3 runs - error bars are not shown as the standard deviations were typically less than $< 1\%$, with the largest being 2.5%. Different test systems are used for each supported MD code in order to avoid making direct comparisons between the performance of the MD codes themselves, and also to illustrate the versatility of MIST to be able to cope with differing force fields, periodic boundary conditions and constraints schemes.

4.3.1 NAMD-Lite

To measure the performance overhead of using MIST with NAMD-Lite, an isolated deca-alanine molecule was simulated using the input settings supplied in the `demo` directory of the NAMD-Lite distribution (also in the `examples/namd-lite/alanin`

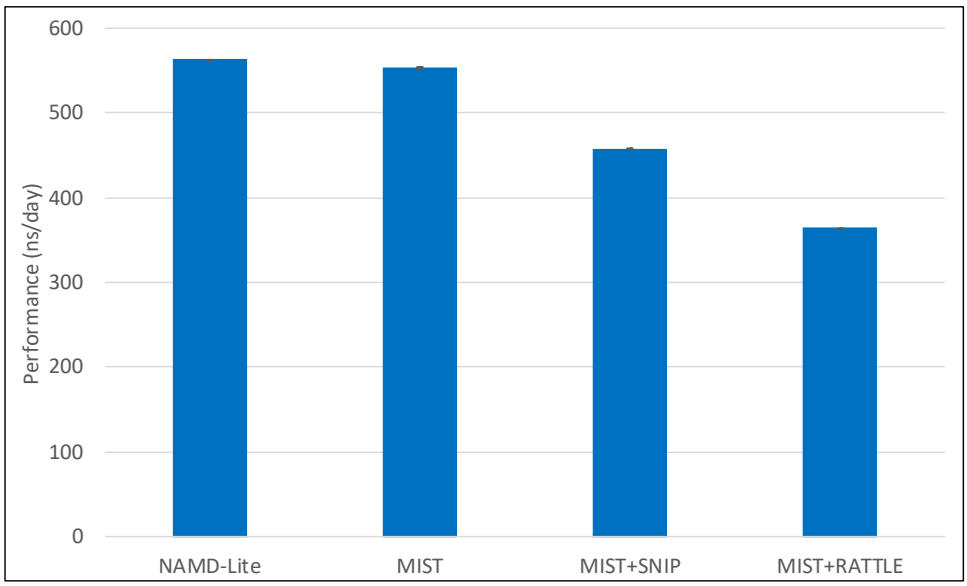


Figure 4.10 *Performance of NAMD-Lite with and without MIST, for a deca-alanine molecule* in vacuo.

directory of MIST). A 12Å cut-off is applied for electrostatic forces, the system is initialised with random velocities at 300K and run for 1 ns of NVE dynamics using a 1 fs time step. A small system is a worst-case test for MIST, since the relatively cheap force evaluation and small number of atoms (66) means that the function call overheads of calling out to MIST to perform the integration are likely to be exposed.

As can be seen in Figure 4.10, only a 2% slowdown is measured when using the velocity Verlet integrator in MIST compared with the native integrator in NAMD-Lite (effectively running with MIST disabled). The figure shows the average over 3 runs for each setting - variability between runs was negligible (standard deviation $< 0.1\%$ in all cases). The performance when constraints are enabled in MIST, using both the RATTLE [10] and Symmetric Newton (SNIP) [19] methods, using the default constraint tolerance of 10^{-8} is shown. Resolving the constraints takes a significant amount of time, with RATTLE reducing the overall performance by 34%. SNIP is significantly faster, with only a 17% performance drop. Direct comparison with NAMD-Lite's constraints implementation is not possible since it only supports the limited capability of SETTLE [153] for rigid water models.

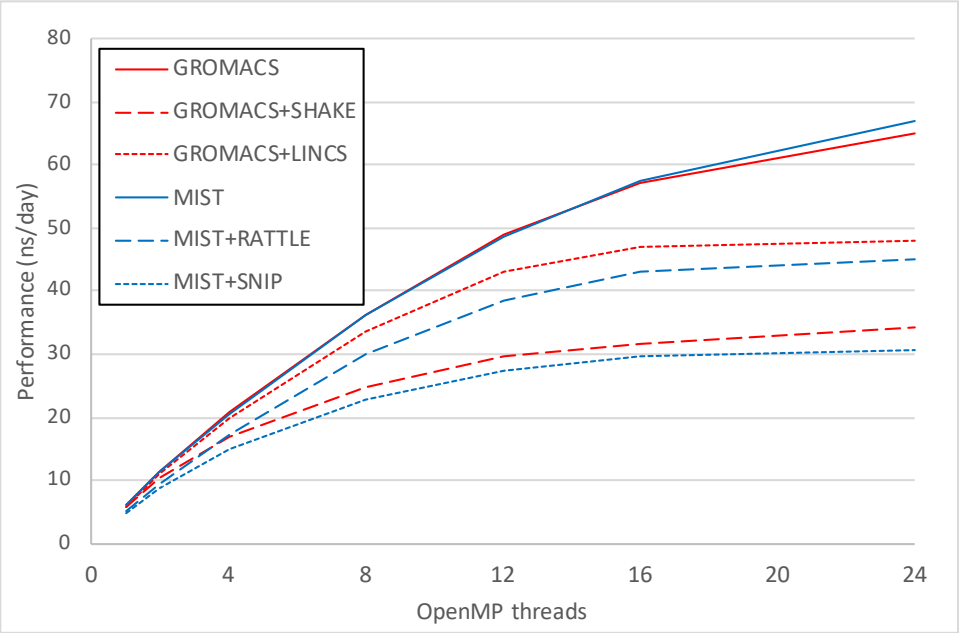


Figure 4.11 *Performance of GROMACS on ARCHER with and without MIST, for the 12,207 atom water system.*

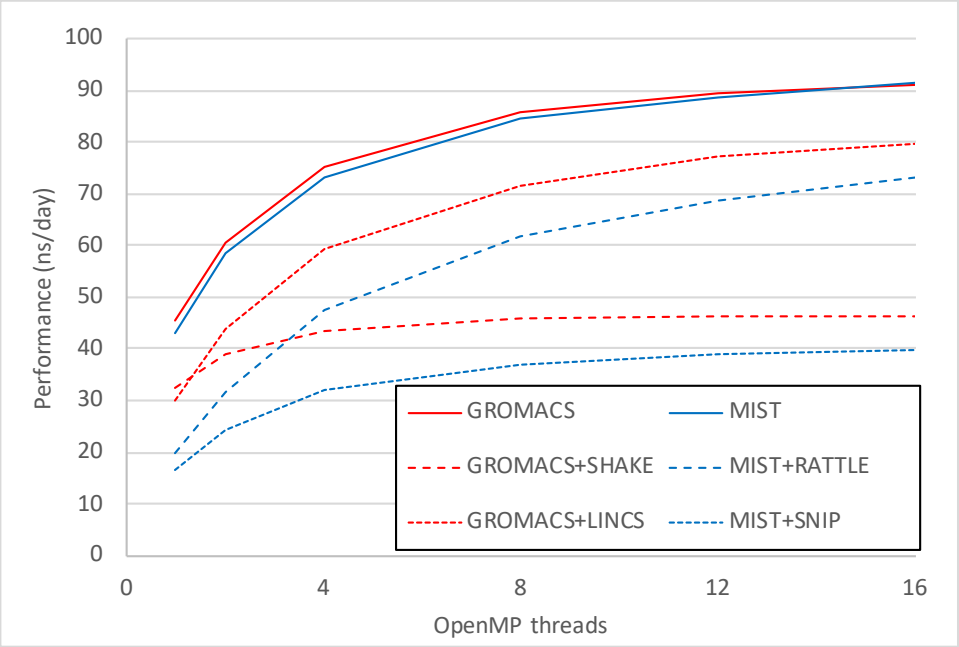


Figure 4.12 *Performance of GROMACS on an NVIDIA K40m GPU with and without MIST, for the 12,207 atom water system.*

4.3.2 GROMACS

The performance of MIST with GROMACS on a single compute node is illustrated using a more realistic-sized system - a 50Å cubic periodic box containing 4069 water molecules. The input geometry and settings are supplied in the `examples/gromacs/water` directory of the MIST distribution. The TIP3P water model from the CHARMM27 force-field is used with a 12Å cut-off for the electrostatic and van der Waals forces. The system was initialized with random velocities at 300K and run for 25 ps of NVE dynamics using a 1 fs time step. For this system (using a single CPU core) 96% of the run time is spent in force calculation and neighbour list search and less than 3% in the integration itself (the `Update` time reported by GROMACS). Both a fully flexible water model and one with bond constraints applied were tested. GROMACS supports the SHAKE and LINCS[100] schemes for resolving constraints and default settings were used for both. In MIST, a relatively loose constraint tolerance of 10^{-4} is set, matching the SHAKE tolerance used by GROMACS.

Figure 4.11 shows the performance achieved for each case using up to 24 OpenMP threads. For the unconstrained case, MIST is within 1% of native GROMACS performance and on 24 threads outperforms GROMACS by 3%. Comparing the constraint implementations, the LINCS[100] algorithm in GROMACS performs best although it is not possible to directly compare it with the SHAKE, RATTLE or SNIP solvers as it does not use a relative constraint tolerance, but rather a fixed (4th) order expansion and a fixed number of iterations (1). Interestingly, for the same tolerance, the RATTLE algorithm implemented in MIST is 12% slower than GROMACS' SHAKE implementation when running on a single thread, but when using 24 threads is 36% faster. Unlike for the small deca-alanine system, SNIP is the slowest algorithm due to the expensive inversion of the gradient matrix. It is important to recognize that SNIP is most advantageous when treating large macromolecules with high bond connectivity, whereas waters are actually better handled by RSHAKE [120] or SETTLE [153].

On the GPU (Figure 4.12), we see a slightly higher overhead between MIST and the native GROMACS Verlet integrator, of around 5.5% using a single core, becoming negligible on 16 cores. This reflects the fact that as the force evaluation using the GPU is much faster, with overall performance of 45 ns/day compared with 6 ns/day on the CPU only, the additional cost associate with calling out to MIST to do the integration is proportionally higher. However, the

use of OpenMP within the MIST integrator offsets this at higher thread counts. Similarly to the CPU, the GROMACS LINCS implementation is fastest, but the difference in performance between the GROMACS SHAKE and MIST RATTLE implementations is much higher, with MIST being 39% slower on a single thread, but 57% faster using 16 threads.

To demonstrate the performance of MIST when run in an MPI parallel environment, a system consisting of the well-known [140, 214] NTL9(1-39) protein is used, which consists of 636 atoms and is solvated in 4488 water molecules, for a total of 14100 atoms in a 50Å cubic periodic unit cell. Electrostatic forces were computed using the PME method, with a short-range and van der Waals cutoff of 1nm. 10,000 time steps of 0.5 fs were run, and the reported performance in ns/day up to 144 MPI processes (6 ARCHER nodes) is shown in Figure 4.13. The performance on a single node using OpenMP parallelisation is also shown for reference. Up to 24 cores (within a single node), the performance impact of using MIST compared with native GROMACS performance is up to 1.5% for OpenMP and up to 6% for MPI. This is attributable to the fact that there are additional MIST calls required whenever GROMACS makes a load balancing step. Beyond a single node, the performance of both GROMACS and MIST starts to become more variable as the inter-node communication cost becomes important, and varies from run to run. Nevertheless, MIST and GROMACS give similar performance up to 96 cores, when the system size is too small to support further strong scaling.

4.3.3 Amber

To test the performance of MIST with Amber the same NTL9 system is used as for GROMACS, but using the CHARMM22 force field with a TIP3P water model and a cut-off of 9Å for real-space part of the electrostatic forces and the long-ranged electrostatics computed on a 54^3 PME grid. The system was initialised with random velocities at 300K, and run for 25ps of NVE dynamics using a 1 fs time step with the `pmemd` or `pmemd.cuda` program. Input files are available in `examples/amber/ntl9` in the MIST distribution. Amber supports bond constraints for hydrogen atoms only on the GPU, so the corresponding `h-bonds-only` setting in MIST is used with a relative tolerance of $1E-5$, matching the default Amber SHAKE tolerance.

Figure 4.14 shows the performance achieved running on a single CPU core on

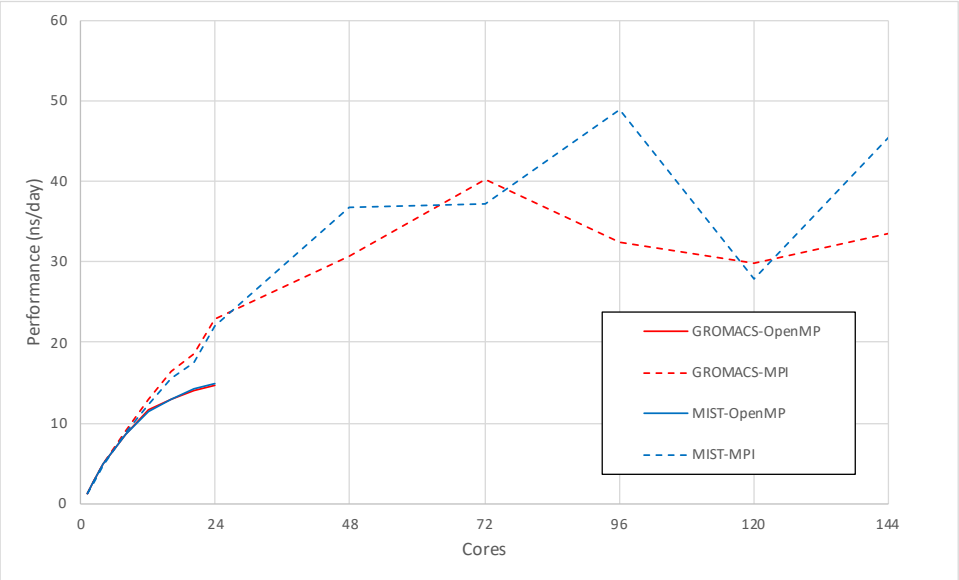


Figure 4.13 *Parallel performance of GROMACS on an ARCHER with and without MIST, for the NTL9 system.*

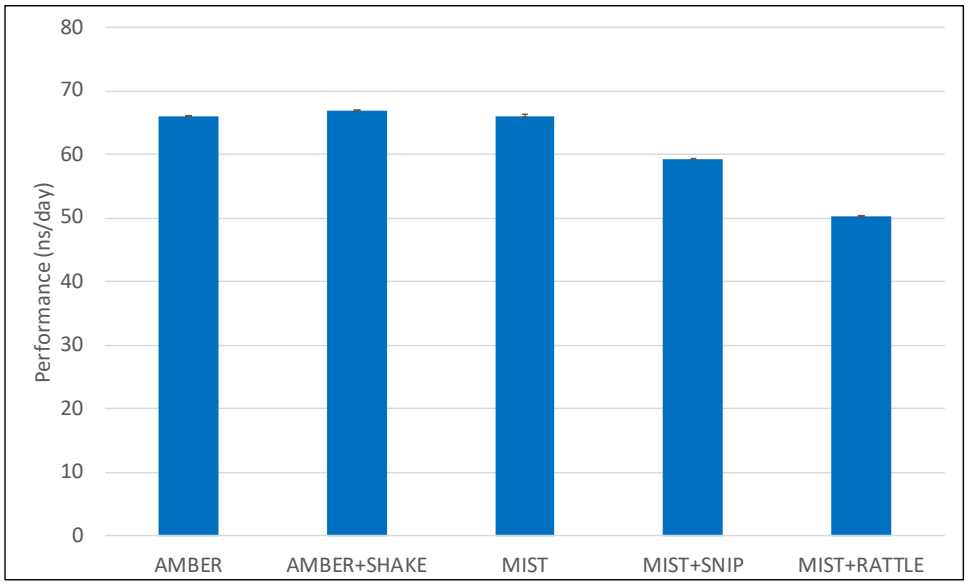


Figure 4.14 *Performance of Amber on ARCHER with and without MIST, for the solvated NTL9 system.*

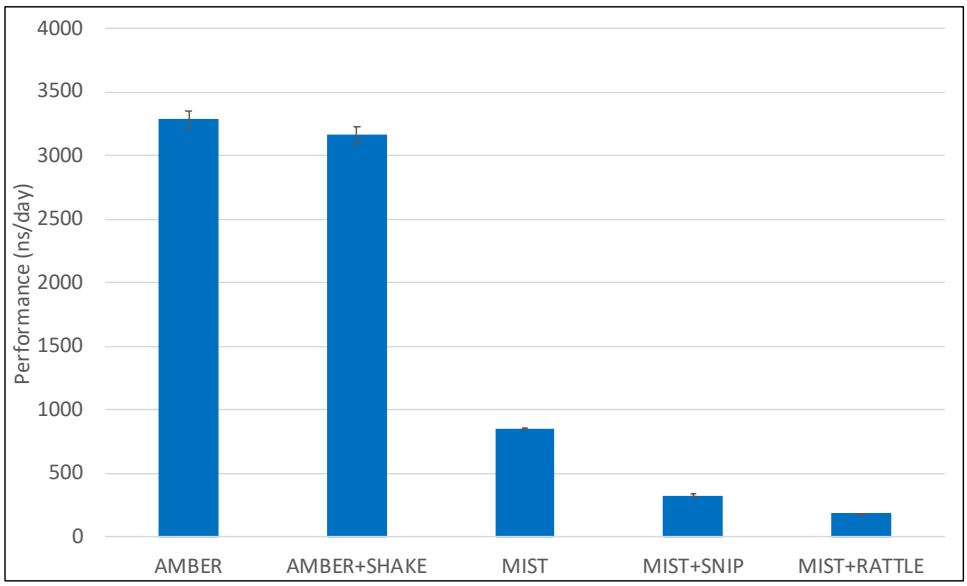


Figure 4.15 *Performance of Amber on an NVIDIA K40m GPU with and without MIST, for the solvated NTL9 system.*

ARCHER (Amber 14 does not have thread parallelisation). It is clear that using MIST has negligible impact on the performance. For the constrained runs, Amber is slightly faster (by 1%) as it is possible to skip the computation of the forces caused by the constrained bonds (`ntf=2` in Amber), offsetting the additional cost of the SHAKE algorithm. Similarly to NAMD-Lite and GROMACS, both the MIST constraint solver algorithms have an additional performance overhead. For this system, SNIP is faster with an 11% drop, compared to 24% for RATTLE.

For Amber, the overhead of using MIST with `pmemd.cuda` is much higher (see Figure 4.15). Whereas GROMACS achieves around 10× speedup with a K40m GPU compared to a single CPU core, Amber achieves a speedup of over 50× by doing the entire calculation (both the force evaluation and integration) on the GPU, thus avoiding relatively high-latency transfers between the GPU and CPU memory. In order to use MIST the updated coordinates must be transferred to the GPU and the resulting forces transferred back again *at each time step*, effectively throttling the GPU by memory transfers. As a result, running with MIST achieves only 844 ns/day, compared with 3282 ns/day using the native integrator running on the GPU. This is still over 10× higher than the 66 ns/day achieved by Amber and/or MIST running on the CPU. As expected, the constrained runs are slower still with a 62% overhead for SNIP and 78% overhead for RATTLE. These overheads are higher than observed in the CPU runs as the force evaluation is much faster so the time spent in the constraint solver is proportionally larger.

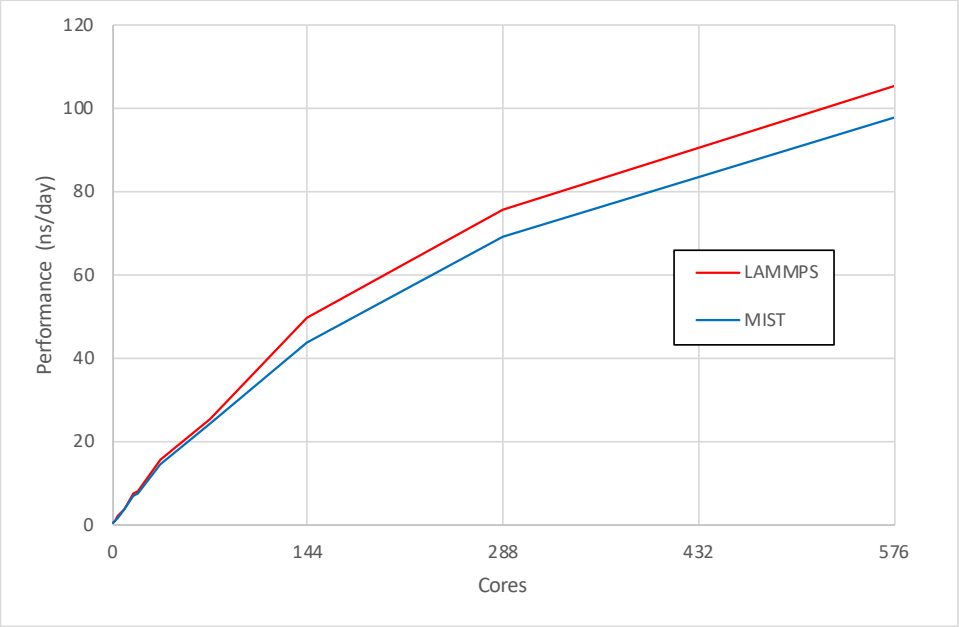


Figure 4.16 *Performance of LAMMPS on Cirrus with and without MIST, for the rhodopsin system.*

4.3.4 LAMMPS

Performance of MIST with LAMMPS is tested with a system consisting of the rhodopsin protein in a lipid bilayer, for a total of 32,000 atoms. Forces are calculated using the CHARMM force-field with a 10 Åcut-off and fully flexible bonds. NVE dynamics are run for a total of 200 2fs time steps, using either the LAMMPS `fix nve` or MIST `VerletIntegrator`. This is an adapted version of the standard LAMMPS `rhodo` benchmark and input files are available in `examples/lammps/rhodo`. Figure 4.16 shows the performance achieved by LAMMPS with and without MIST up to 576 cores, where performance starts to drop off. The overhead of using MIST varies from 5% on 8 cores, up to a maximum of 13% on 144 cores. As well as the serial overhead of calling in to MIST, the majority of the difference is due to increased force evaluation cost. Since MIST only does load balancing at the start of the step rather than at the point where forces are calculate, it is necessary to slightly increase the size of the halo (or 'skin' in LAMMPS terminology - see Section 2.1.4) between processes, resulting in increased communication cost. For example on 144 cores, the average number of atoms in the halo region on each process is 5137 without MIST rising to 7082 with MIST (a 37.9% increase) with a commensurate increase in communication time by 47.2%.

4.4 Discussion

In this chapter, I have shown that MIST gives correct results with respect to equivalent integrators implemented in each of the supported MD codes. Where no direct comparison can be made I show that the integrators perform as expected and so MIST is reliable enough to be useful as a platform for novel algorithm development, without requiring care to be paid to the implementation details below MIST's abstraction layer. No attempt has yet been made to cross-validate the same integrator running in MIST across two or more MD codes. In practice, even for a well-defined choice of molecular system and force-field, it is difficult to assess whether differences in results are due to unavoidable numeric effects or actual errors! It is likely that the best approach would be to carry out production quality calculations, aiming to demonstrate agreement between some converged ensemble average property rather than direct agreement between energies, for example.

In Section 3.1 three design principles were laid out which all of the subsequent design and implementation choices were intended to support. The validation tests, performance results, and range of integrators implemented using MIST can be used to assess the extent to which these principles have been upheld.

- **Expressiveness**: I described in Section 3.3 eleven different integrator schemes ranging from simple, classical examples such as Verlet and Runge-Kutta to much more complex extended variable and tempering schemes. Some of these were implemented by students and postdoc collaborators with little need for support. In cases where I collaborated with other developers, this was mainly to improve the performance of their implementation. This demonstrates the fact that MIST is easy to develop on top of, is expressive enough to permit a wide range of different algorithms, and that developers do not need to concern themselves with details such as performance optimisation - although this can still be added later, if desired. However, the fact that MIST is implemented in the context of the standard MD loop of the host code makes it impractical to implement static methods such as geometry optimisation in MIST - any calculation run with MIST necessarily produces a time-series trajectory. In addition, MIST does not have access to second derivatives of the energy, which are usually needed for such algorithms.

- **Performance**: The performance tests above show that, except in the case of Amber on GPU, using an integrator from MIST has only a small overhead compared to a native integrator - typically a few percent when running on a single core, rising to around 10% for parallel runs. Performing constrained integration comes at an additional overhead with the cost depending on the system size, topology and the constraint solver method and accuracy chosen, but is comparable to common methods such as SHAKE implemented in Amber and GROMACS. I have shown that MIST can take advantage of all three of the most common types of parallelism - OpenMP threading, MPI distributed memory parallelism and GPU acceleration - without exposing the complex parallel code to the algorithm developer. I argue that the small performance overhead of using MIST is a reasonable price to pay for the increased expressiveness and reduced complexity of using MIST's abstraction compared to implementing directly in existing codes and so does allow new algorithms to be demonstrated at scale on real-world problems with production quality force-fields.

- **Portability**: I have demonstrated MIST working correctly with four different MD codes, covering Fortran, C and C++. Each of the integrators in MIST may be used with any of the supported codes - with no changes to the integrator source code required. A range of example inputs, particularly for the Langevin Dynamics integrator, are included with MIST - thus broadening the availability of the scheme well beyond its only existing implementations in NAMD (where it is not even documented),and recently, in OpenMM. As shown, the MIST input file is also very simple and code-independent and only a single new parameter is introduced to the host code's input file to simply turn on MIST usage and so a complete division of responsibility is achieved between MIST's input file which configures the integrator and the host code's input which covers the forcefield, I/O, observables, and run control. In addition, while it was not an explicit aim originally, the ability to rapidly add support for additional codes is also a significant benefit and is shown by the fact that I was able to add support for Forcite - an *a priori* unseen code-base - within a few days work at BIOVIA.

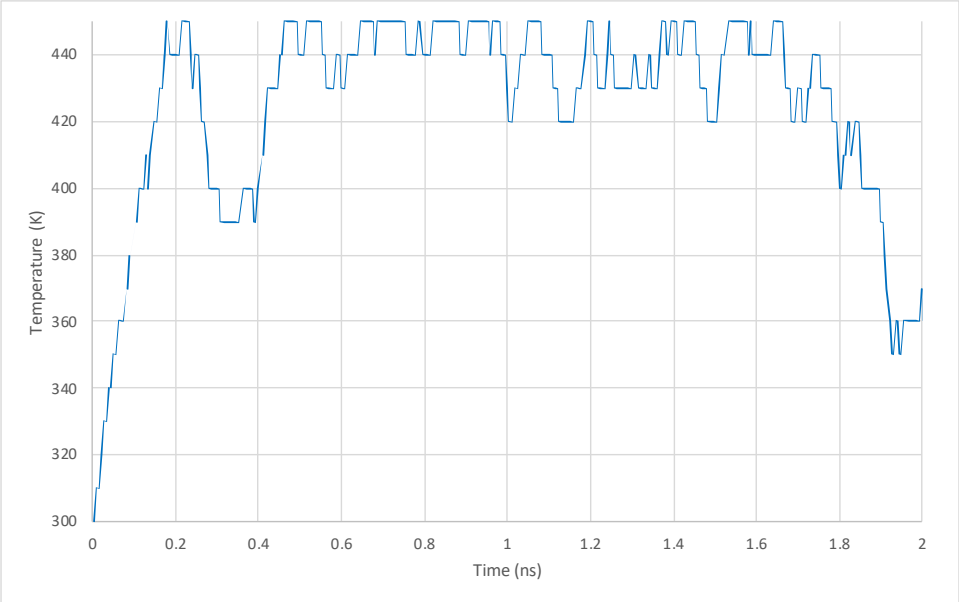
Chapter 5

Application: Simulated Tempering of Alanine-12[1]

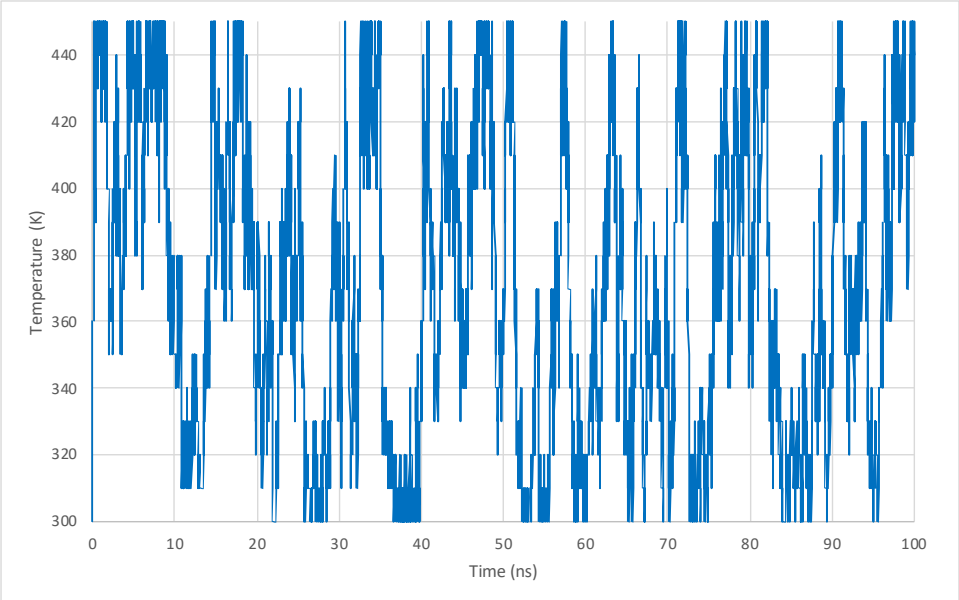
Alanine-12 is a classic example of an α-helical biomolecule. It is particularly interesting to disentangle the effects of solvation and temperature on the unfolding process. At room temperature, the unfolding process cannot be simulated directly with standard Molecular Dynamics because of slow kinetics. The MIST implementation of Simulated Tempering (see Section 3.3.6) is used to sample the free energy landscape of the alanine-12 molecule, previously studied using the Diffusion-Map-directed-MD method [176]. Starting from the helical configuration *in vacuo*, a total of 1 μs of MD was run using a 2 fs time step. The Amber96 forcefield was used with a 20Å cut-off for electrostatics, constraining all bonds using the SNIP method (see Section 3.3.8). Temperature was controlled using Langevin dynamics ($\gamma = 1.0ps^{-1}$), either set to 300K to sample an NVT ensemble or varied using Simulated Tempering with temperature states ranging from 300K to 450K at 10K intervals.

Figure 5.1a shows that the Simulated Tempering algorithm visits all temperature states within the first 200ps, and the weights $f_0..f_{15}$ remain largely unchanged throughout the rest of the rest of the run (Figures 5.1c and 5.1d). Figure 5.1b shows that the temperature varies randomly between states during the remainder of the run and so the configurations generated can be used to compute free energies. Free energies at a specific temperature T can be obtained by selecting from the trajectory the configurations obtained at the corresponding

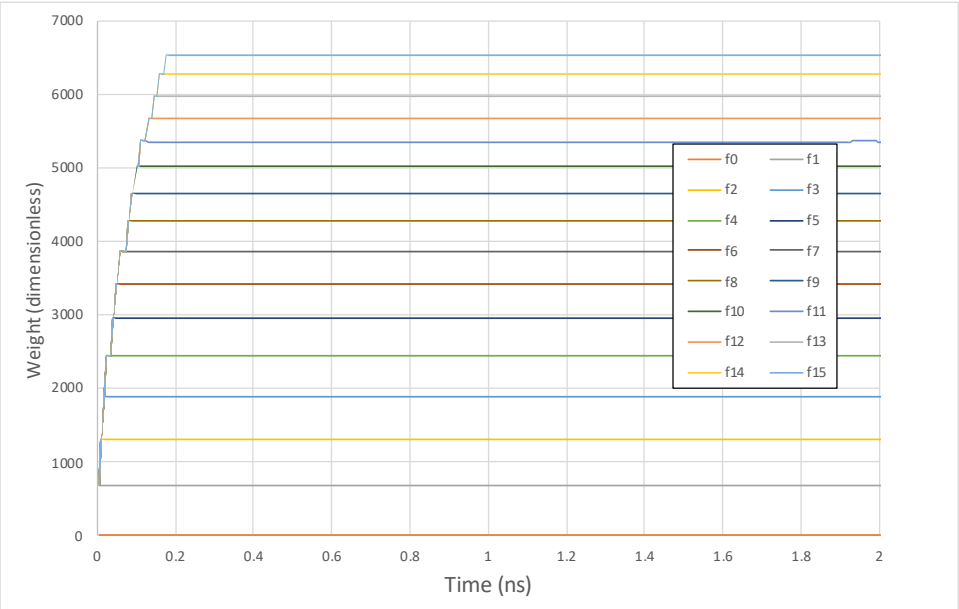
[1]This chapter is a more detailed description of the application first published in [24].



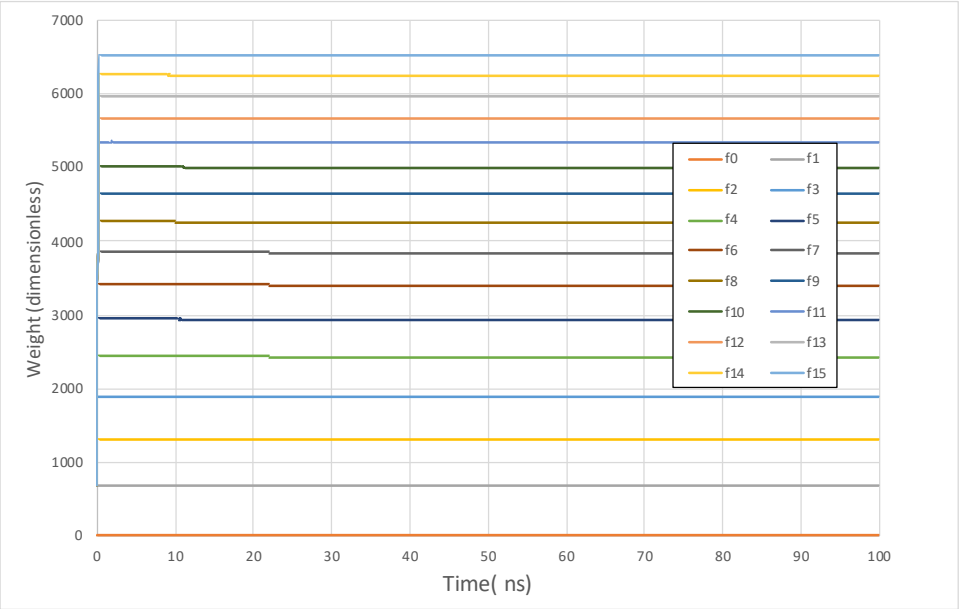
(a) *Temperature state during first 2ns.*



(b) *Thermostat state during first 100ns.*



(c) *Weights during first 2ns.*



(d) *Weights during first 100ns.*

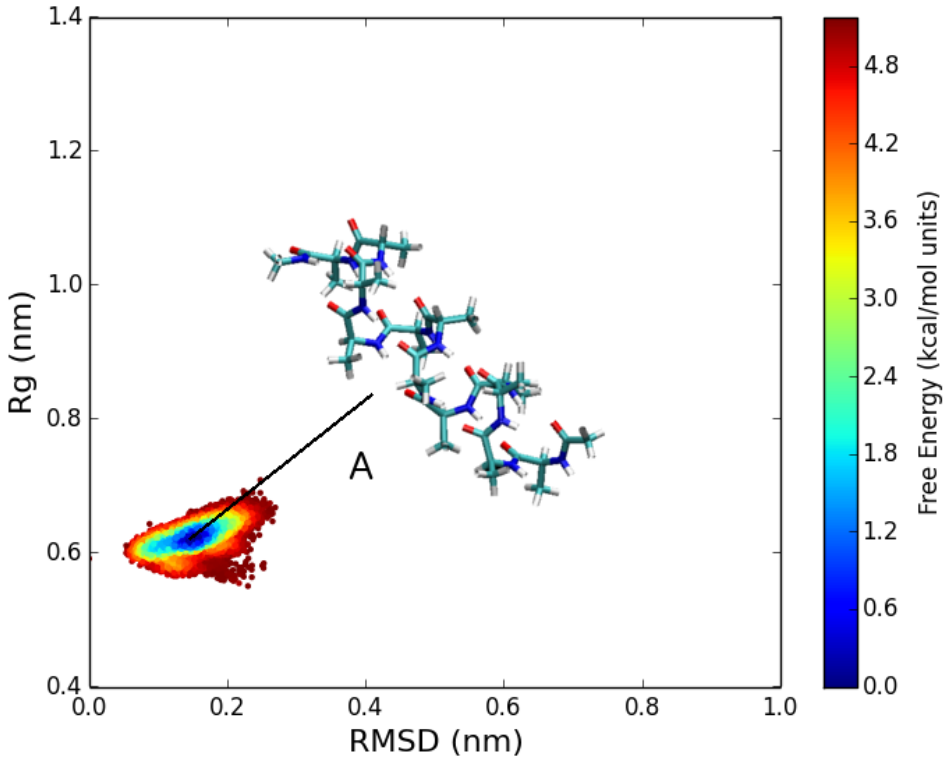
Figure 5.1 *Simulated Tempering temperature states and weighting over time.*

temperature state. By definition the probability of the system occupying a given microstate is $\propto e^{-F}$ and so the free energy is can be computed by binning configurations according to some variables and taking the negative log of the number of configurations in each bin. Here, the free energy surfaces are plotted as a function of RMSD from the initial state and the radius of gyration R_g in Figure 5.2. The RMSD gives an indication of the global dissimilarity to the reference (helical) conformation and R_g a measure of the compactness of the structure around its principal axis. Both can be computed by the GROMACS `g_energy` utility.

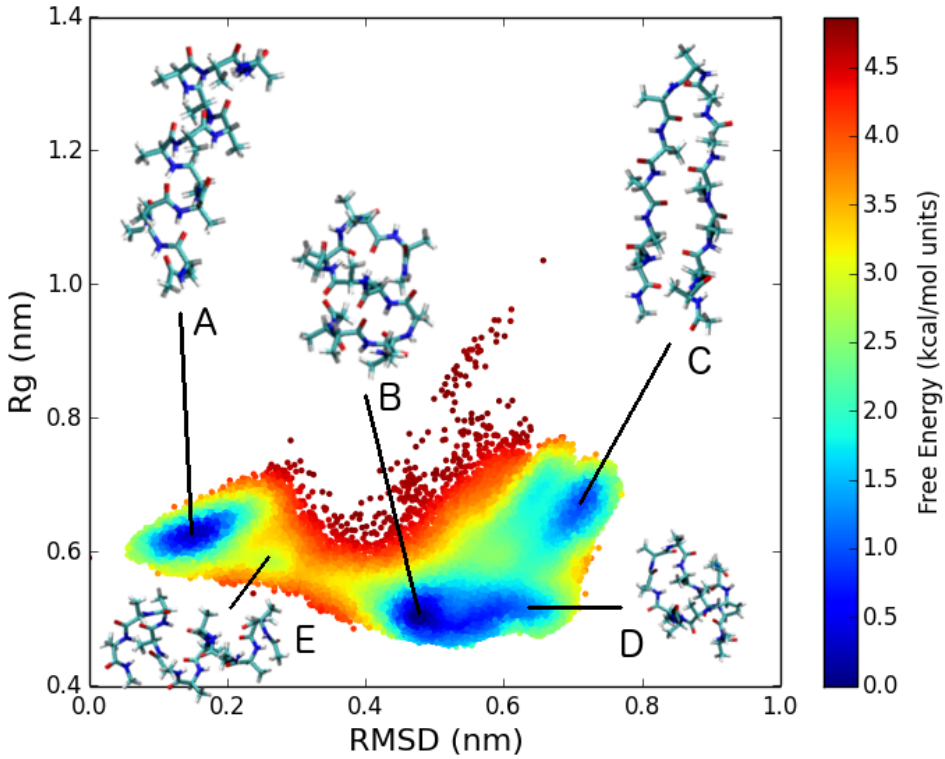
As expected, Figure 5.2a shows that at 300K, the system is trapped in a local minimum around the helical state (labelled A). Using Simulated Tempering, the elevated temperature is enough to allow the system to explore into a wider range of (partially uncoiled) configurations - comparable to those accessed by plain MD at 400K in Figure 7 of [176].

Figure 5.2b shows the complete set of configurations sampled, including those at temperatures greater than 300K. Restarting simulations from the configurations labelled B (a compact structure consisting of three hairpin turns) and C (where the two termini are aligned and a complex twisted structure forms in the backbone) and running a subsequent 1 μs of NVT dynamics at 300K shows that configuration B exists in a stable minima (Figure 5.2c), whereas the system is free to migrate between configurations C and D via a transition state F (Figure 5.2d), where the termini of the molecule have turned back on themselves.

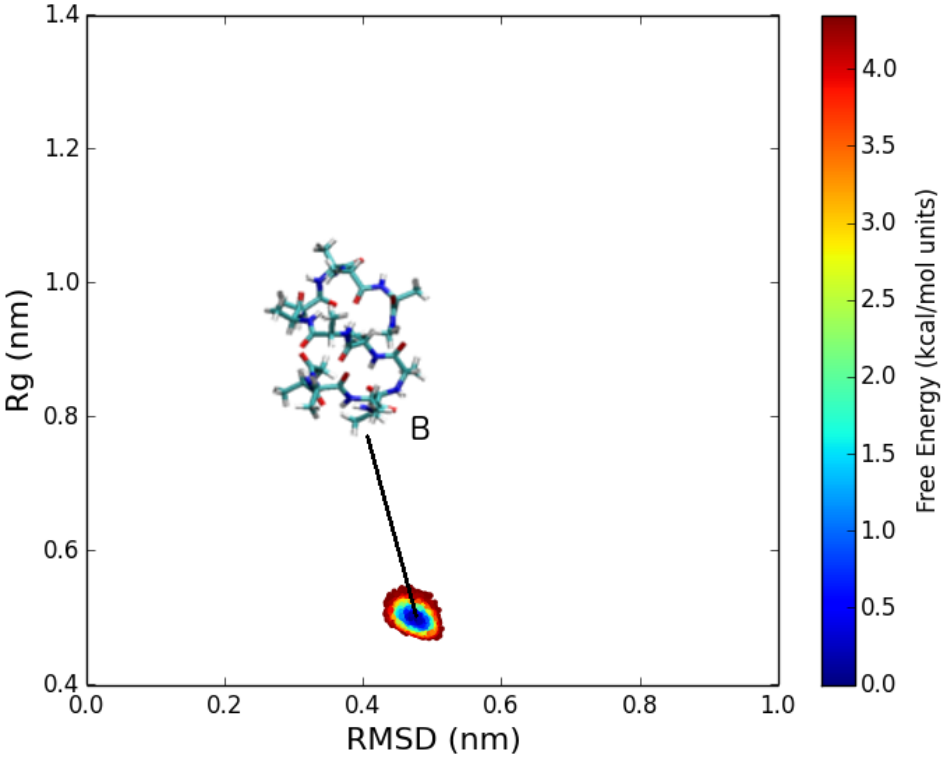
In contrast, even at 300K (Figure 5.3a) the solvated system is able to access a much wider range of states including the fully unfolded state (H) and a large basin (G) containing various extended structures. This is qualitatively similar to the behaviour of deca-alanine observed in [97], which has extended conformations of comparable free energy to the helical state. Physically, the addition of water molecules provides an alternative hydrogen bonding route than can effectively 'bridge' between -CO and -NH groups in the backbone, stabilising extended structures that are not observed in the *in vacuo* ensemble. Compared with the *in vacuo* simulations, the molecule does not sample the compact ($Rg \simeq 0.5$) states B and D, but instead states like I and J where both ends of the molecule are unbound and a hairpin turn or complete helix is present in the middle. Since the energy barriers between states are relatively low (< 2.8kcal/mol) compared with barriers of up to 4 kcal/mol in the vacuum case, MD at 300K is able to access all states and so Simulated Tempering does not provide any access to any



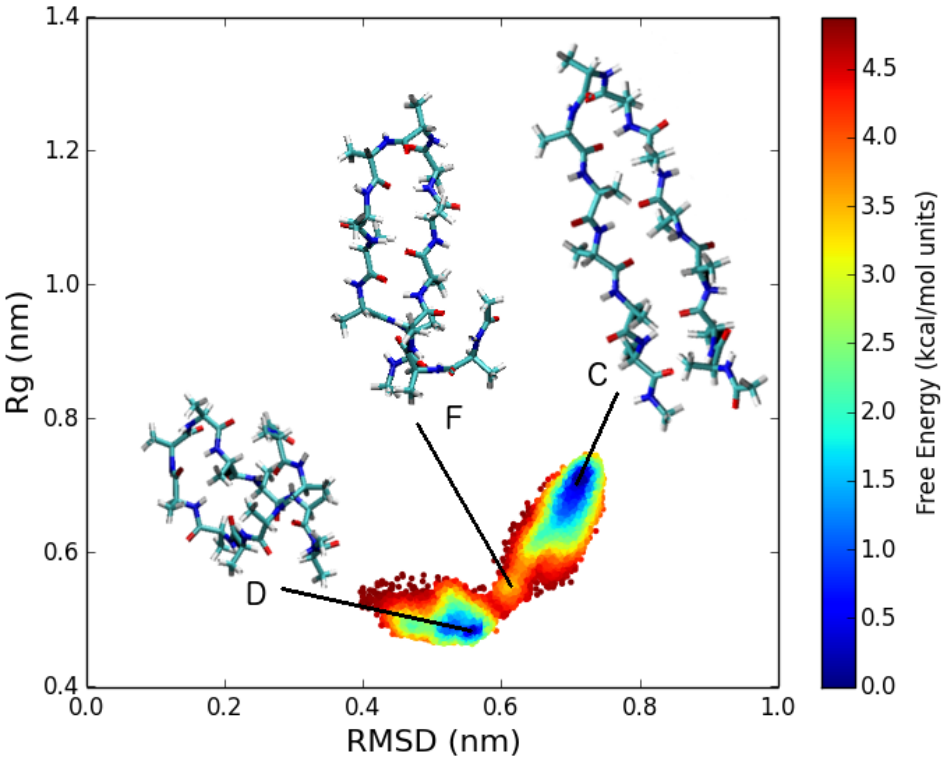
(a) *NVT at 300K, starting from helical configuration.*



(b) *Simulated Tempering 300-450K.*

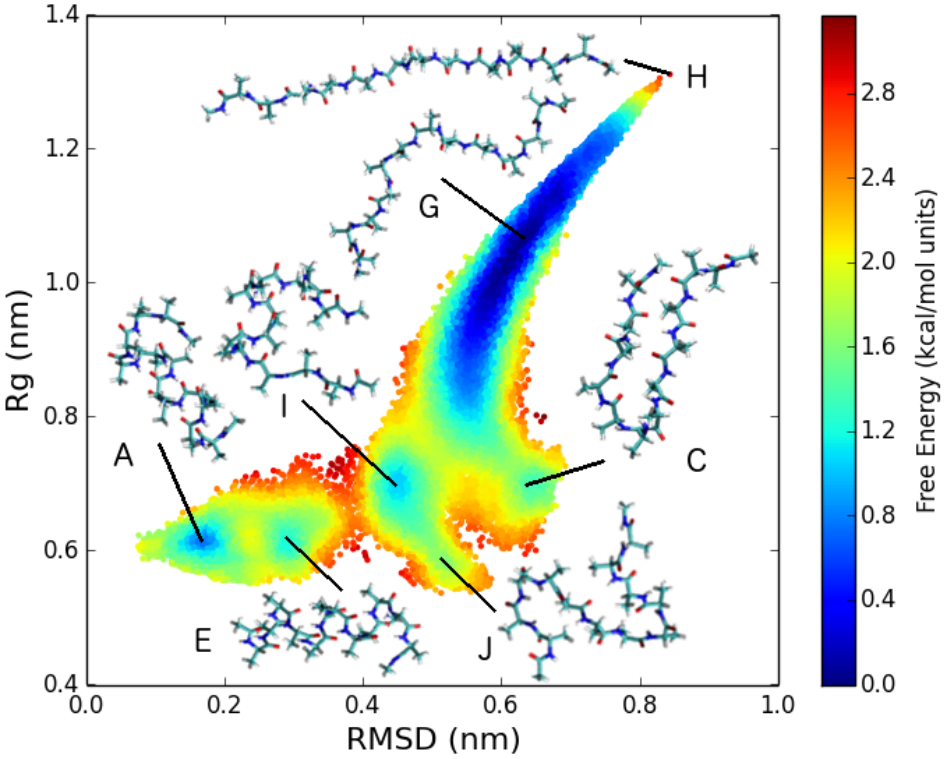


(c) *NVT at 300K, starting from configuration B.*

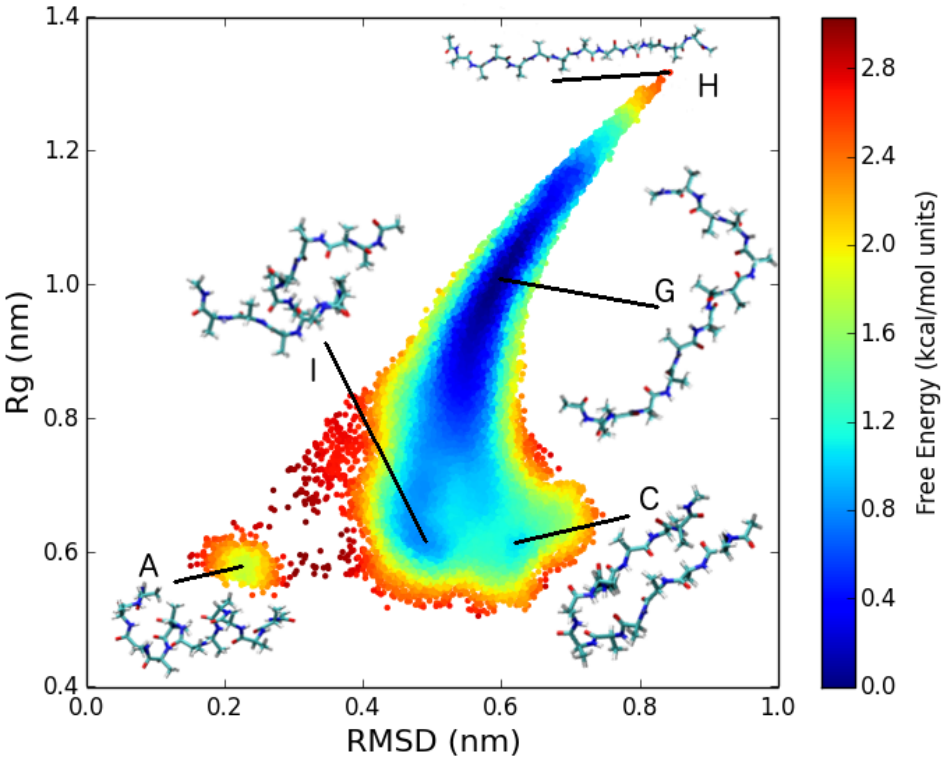


(d) *NVT at 300K, starting from configuration C.*

Figure 5.2 *Free energy surfaces of Alanine-12* in vacuo.



(a) *NVT at 300K, starting from helical configuration.*



(b) *Simulated Tempering 300-450K.*

Figure 5.3 *Free energy surfaces of Alanine-12 solvated in TIP3P water.*

qualitatively new states (Figure 5.3b) in this case.

For this application, using MIST to run Simulated Tempering is much more convenient than the pre-existing scripts for GROMACS [220]. Running the entire calculation as a single job avoids the need to repeatedly start and stop (very short) GROMACS calculations which could have a significant effect on the overall run time, especially if each step has to be launched on a parallel machine. Analysis of a single, long trajectory is also simpler as there is no need to concatenate multiple short trajectory files. Finally, as mentioned in Section 3.3.6, the use of the Langevin BAOAB thermostat is expected to produce a more accurate sampling of the NVT ensemble than standard thermostats.

Chapter 6

Application: Properties and Structure of Olivine Melts

Geophysics, the study of the Earth system, has developed a complex set of models to describe the layered structure of Crust, Upper and Lower Mantle, and Inner and Outer Core [11] in the Earth's interior. The mean physical conditions of temperature and pressure at various depths are described by models such as the Preliminary Reference Earth Model (PREM) [64]. However, these models ignore a number of dynamical processes such as turbulent convection in the Mantle and Core, heat transfer across boundary layers, and subduction and extrusion of the crust. Any such global model is underpinned by knowledge of the chemical composition of the various regions and the resulting material properties which are used to parameterise to model.

Knowledge of material properties can be derived from experiment, and there is a large body of data developed over the last 30 or more years (for example [37, 56, 98, 149]) using mechanical measurements and spectroscopic techniques. In addition, computer simulation has an important role to play, firstly for extreme conditions of temperature and pressures which cannot currently be accessed by experiment [197], secondly to made predictions which may be later tested experimentally [7], and thirdly to provide an atomistic rationale for observations [118].

The principal computational technique used is Molecular Dynamics, with forces derived either from an empirically-fitted classical force-field or from Density Functional Theory (DFT). MD calculations are computationally expensive,

especially when system sizes containing 100s or 1000s of atoms are modeled using DFT. To compute long enough trajectories to get a good sampling of the material at equilibrium requires the use of efficient software such as CP2K [105, 127] running on modern High Performance Computing systems.

This chapter documents the development and testing of accurate and efficient models of minerals such as olivine which make up the majority of the Upper Mantle, and the use these models to provide atomistic-scale insights into their observed structure and physical properties.

6.1 Earth Structure

The modern concept of the Earth as an approximate sphere has been held since Pythagoras in the 6th Century BC. While astronomical observations by the Greeks some 300 years later lent physical evidence to this, the current understanding of the Earth as a complex and dynamic system is a relatively recent innovation. In the 18th Century, James Hutton proposed [106] that the observed rock formations on the Earth's surface are a result of processes long ago and at great depth and moreover that these processes are still ongoing, albeit at a slow pace spanning millions of years. This idea was further developed into the theory of continental drift (e.g. by Wegener [216]), which explained not only the obvious interlocking shapes of South America and Africa, but also the discoveries of similar fossils and mineral deposits on both Atlantic shores. Continental drift lacked a sound underlying physical mechanism, and it was not until the middle of the 20th Century that this was resolved by the theory of Plate Tectonics [32, 217] - that Earth's Lithosphere (the outermost mechanical layer, consisting of the crust and upper mantle) is formed from multiple plates which move atop the lower mantle.

The established models of the Earth are thus relatively recent, and still under constant revision. Fowler [73] presents a simplified model, with an outer Crustal layer of silica-rich rock, only 10s of kilometres thick. Beneath this lies the Mantle, a 2900km depth of primarily magnesium silicate melts, and in the centre, an iron core, of which the outer 2200km are liquid and the inner 1200km are solid. Anderson [11] presents the more nuanced view of Bullen [44] with no less that 8 discrete regions (see Table 6.1). It is clear from this classification that not only are there transition regions rather than clear boundaries, but that discontinuities

Region	Depth (km)	
A	0 - 33	continental crust
B	33 - 410	upper mantle
	220	Lehmann discontinuity
C	410 - 1000	transition region
	650	discontinuity
D'	1000 - 2700	lower mantle
	1000	Repetti discontinuity
D"	2700 - 2900	transition region
E	2900 - 4980	outer core
F	4980 - 5120	transition region
G	5120 - 6370	inner core

Table 6.1 *Bullen's regions of the Earth's interior, adapted from [11].*

in physical properties (e.g. density, seismic wave velocity) exist within the regions themselves!

6.1.1 Physical Properties and Composition

The upper mantle in particular does not lend itself to a simple characterisation. This complexity is shown in the Preliminary Reference Earth Model (PREM) of Dziewonski and Anderson [64], a 1-dimensional (radial) model of density and seismic wave velocity, where the entirety of the earth is considered isotropically except for the uppermost 200km. The physical quantities in the model are represented by polynomials which are fitted to available seismological data. A number of later models have superseded the PREM, based on more recent data and improved mathematical models of seismic wave propagation, notably IASP91 (1991, includes wave velocities) [115], AK135 (1995) [116] and STW105 (2008) [124]. Unlike the other parameterised models, AK135 uses a discretisation scheme and interpolation must be used to derive values at intermediate points. However, as is clear from Figure 6.1 all of the models are closely matched, varying only in particular areas of detail around region boundaries.

Given the density data from any given model, one can compute the hydrostatic pressure by integration of:

$$\frac{dP}{dr} = -\rho g$$

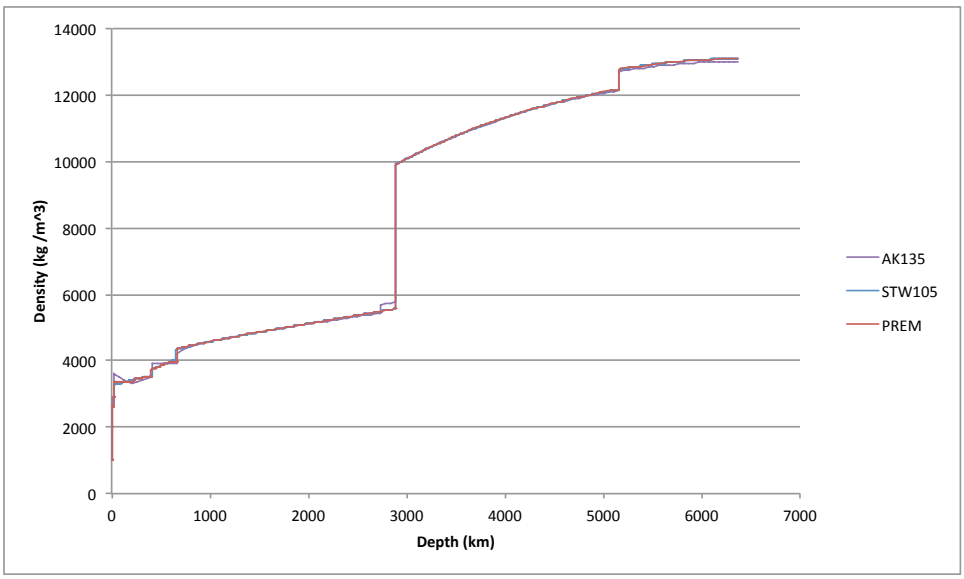


Figure 6.1 *Comparison of densities predicted by PREM, AK135 and STW105.*

where r is the radial distance from the centre of the earth rather than the depth, ρ is the local density and g the gravitational constant. Figure 6.2 shows representative pressure data calculated from the PREM density tables, as well as indicative temperatures. Definitive temperature data at various depths is harder to obtain, since it is typically inferred from melting temperatures of the constituent materials at the relevant pressures, often combining data from experiment and computer simulation. However, the temperature does increase monotonically with depth, and a review by Boehler [35] gave temperatures ranging up to 1830K in the Lithosphere, from 1830K to 2600K in the Mantle followed by a discontinuity up to 4000K at the Core-Mantle Boundary (CMB) and 5000K at the Inner Core Boundary (ICB). Subsequent studies [7, 13] have revised the ICB temperature to around 6320K, and suggest the lower value of Boehler was due to recrystallisation of the melt affecting accurate determination of the phase change. Figure 6.2 includes the updated data.

As well as bulk properties determined by seismic wave propagation, it is possible to estimate the composition of the various layers. In the case of the crust this can be done directly via borehole sampling, whereas the composition of the mantle must be inferred from the content of magmas which have emerged and cooled such as Mid-Ocean Ridge Basalts (MORBs) or volcanic magma flows. The composition of meteorites (*cosmochemistry*) also places some constraints on what material may have formed the earth in the early solar system.

Starting with the crust, the primary constituents are plagioclase, a calcium/-

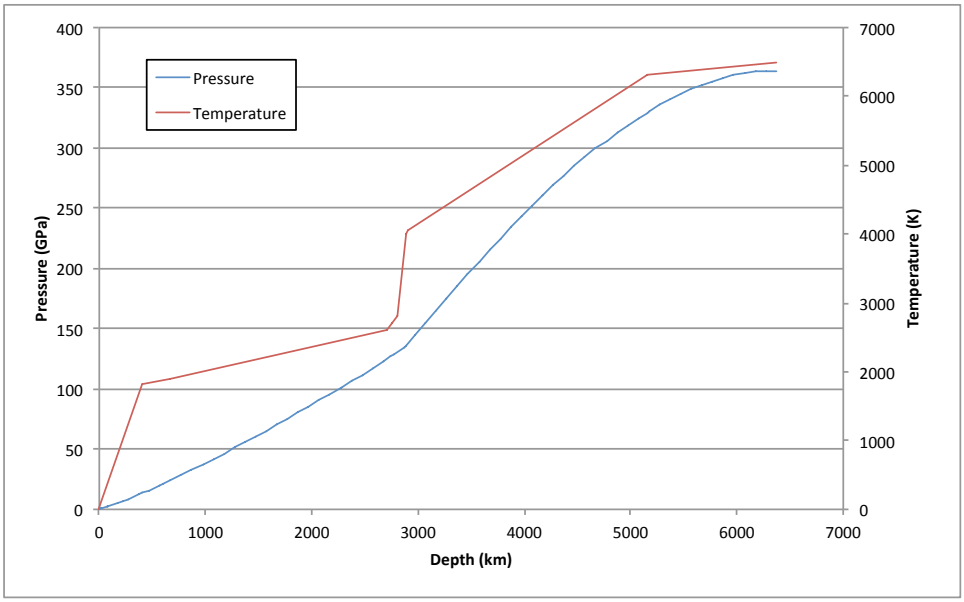


Figure 6.2 *Pressure (from PREM data) and Temperature (from [35] and [13]) against depth.*

sodium tectosilicate solid solution $\{NaAl,CaAl_2\}Si_3O_8$, potassium K-feldspar $KAlSi_3O_8$ and Quartz SiO_2 [11]. The upper Mantle consists of various silicate materials, principally olivine (magnesium and iron silicate) $\{Mg,Fe\}_2SiO_4$, pyroxene $\{Mg,Fe\}_2Si_2O_6$ and garnet $Mg_3Al_2(SiO_4)_3$ [45]. In the Transition Zone, higher-pressure phases of similar minerals occur such as ringwoodite and wadsleyite olivine, and deeper still magnesium and iron perovskites $\{Mg,Fe\}SiO_3$, and Mg/Fe oxide (ferropericlase) $\{Mg,Fe\}O$. The Core is known to have a somewhat simpler composition, being primarily iron, with around 6% by mass of nickel [197] plus trace contributions of both heavier (gold and other transition metals) and lighter (e.g. oxygen) elements.

The properties and composition discussed so far are all averages. In fact the Mantle (and the Core) are dynamic, complex systems containing higher (sinking) and lower (rising) density regions resulting in plumes and convective flows. At plate boundaries, sections of the Lithosphere may be subducted (that is pushed below an opposing plate), and sink into the Mantle.

Nevertheless, one thing that all the models agree upon is that the upper and lower mantle make up the largest proportion of the Earth's interior by mass (around two thirds, depending on exactly how the extent of the mantle is defined) and understanding its behaviour requires input from several disciplines - Seismology, Fluid Dynamics, Materials Science, Minerology, and more - clearly a complex multi-physics problem.

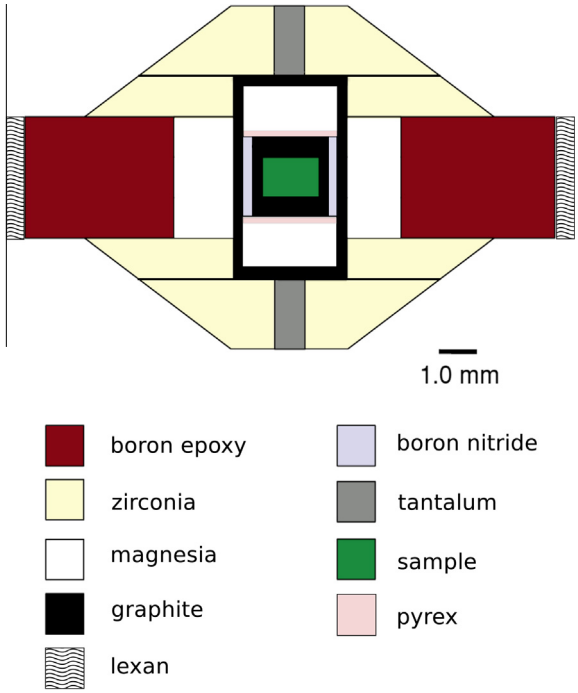


Figure 6.3 *Schematic of the construction of a DAC, from [185].*

6.1.2 Experimental Approaches in Mineral Physics

Experimental data on deep-earth minerals can be broadly understood in two categories: bulk physical properties such as compressibility, density, heat capacity and associated phase diagrams, and structural properties including crystal lattice dimensions, the arrangement of atoms, coordination numbers and the distribution of interatomic distances.

Physical properties may be measured using mineral samples (for example [180]) or synthetic preparations (as in [56]). To explore the phase diagrams of a material the temperature could be controlled using a hot-gas furnace [98, 194] and measured via thermocouples. More recently studies have used current-induced heating in a wire containing a small hole in which the sample is placed [37, 179], and the temperature is calculated based on the measured current flow. An alternative, more modern technique is the use of laser-based heating [185]. High pressures are typically achieved via the use of a Diamond Anvil Cell (DAC), which concentrates the force on the sample contained in a cell (see Figure 6.3) between the tips of two small, incompressable diamonds.

Once the desired experimental conditions are achieved, various apparatus for

measuring properties such as commercial calorimeters may be used. The internal structure of the sample may be probed via a range of different interferometry or spectroscopy techniques, each of which has particular advantages:

- **Fizeau interferometry** is the technique used in [149] where the sample is located between two reflecting plates and illuminated by visible light at a particular wavelength (there 546nm), to generate interference fringes at a detector. Since the fringe spacing depends on the spacing between the reflecting plates, expansion of the sample with heating can be computing from the corresponding shift in the fringe spacing.

- **X-ray diffraction** (e.g. in [37]) involves placing the sample in a collimated (but not necessarily monochromatic) X-ray beam, and measuring the pattern of the elastically scattered photons on a detector. The intensities of the scattered photons gives the Structure Factor, a representation of the reciprocal crystal lattice. Using Fourier transforms and *a priori* knowledge of the chemical composition of the crystal it is possible to reconstruct the real-space structure.

- **XANES** (X-ray Absorption Near Edge Structure) and **EXAFS** (Edge X-ray Absorption Fine Structure) are absorption spectroscopies where the frequency of the incident X-rays are tuned to excite electrons in the material and the absorption at each frequency is measured. Compared to diffraction, X-ray absorption gives more detail on the local structure which can be used compute radial distribution functions, for example [108].

6.2 CP2K and the GPW method

For this work I chose to use CP2K [105, 127], a powerful and scalable program for atomistic simulations of a wide range of systems including condensed phase, molecular systems and complex interfaces. Developed since 2001 by an international collaborative team (including myself), CP2K is freely available under the GNU General Public License from the CP2K web site [202]. Although written in Fortran 95, CP2K is designed from the outset in an object oriented manner to allow easy extensibility and composability of different methods and algorithms. As a result CP2K features a wide range of force evaluation models including classical potentials, Semi-empirical schemes, Kohn-Sham DFT, and

more accurate hybrid DFT-Hartree-Fock [86] and post-HF correlation methods MP2 [57] and RPA [58], as well as allowing arbitrary combinations of these. Built on top of these are many tools including Molecular Dynamics in various ensembles, Monte Carlo, geometry and cell optimisation and Nudged Elastic Band. CP2K consists of around a million lines of code, and with an average of two commits per day to the github repository, development is rapid. To support this a set of over 3300 test input files is used as an automated regression test suite to ensure continued code correctness (see [152] for details), and also to provide examples for users.

There were two principle reasons for choosing CP2K. Firstly, the ability to model a system using both classical and DFT approaches within the same framework makes comparison of the two approaches very straightforward. Secondly, since *ab-initio* MD calculations are relatively expensive, to allow modeling of larger systems requires a very efficient and scalable implementation of DFT. One of the most widely known features of CP2K is QuickStep [211] - also known as the Gaussian and Plane Waves (GPW) method - a dual basis approach to solving the Kohn-Sham equations, where atom-centred Gaussian basis functions are used to represent the wavefunctions, and an auxiliary basis of Plane Waves is used to expand the electronic density and efficiently compute the Hartree energy. The algorithm for transforming between the Gaussian basis stored as coefficients in a sparse matrix and Plane Waves stored on a regular 3D grid makes use of auxiliary 3D real space grids as a means to store the density before the Plane Wave coefficients are calculated using a Fast Fourier Transform. The mapping from matrix elements to the real space grids is referred to a *collocation* and the reverse as *integration*. As a result, the Kohn-Sham matrix (and total energy) can be computed in quasi-linear time - the FFT is $\mathcal{O}(n \log n)$ - and so can easily scale to thousands of atoms/electrons. In addition to QuickStep, CP2K also implements the Orbital Transformation [210] method as an alternative to the traditional diagonalisation approach to wavefunction orthogonalisation during SCF. While still cubically-scaling in the number of atoms, OT has been demonstrated to outperform diagonalisation by a factor of 10 or more for typical systems [210]. The combination of these two approaches gives CP2K excellent efficiency and the ability to simulate large systems within the local DFT approximation. CP2K has been parallelised using MPI and OpenMP [22, 23] and optimised particularly for the Cray XT/XE/XC architectures, and good performance has been demonstrated for relevant system sizes [28].

6.2.1 Simulation Setup Tools

A set of C and Fortran programs has been developed to assist with setting up initial structures, analysis and visualisation of MD trajectory data. Specifically:

- **Lattice transformations**: scaling, shearing, translating and duplicating particle coordinates in an XYZ file format.
- **Trajectory post-processing**: applying minimum image convention to transform all particles into the unit cell for visualisation, preserving connectivity of SiO_4 tetrahedra across periodic boundaries.
- **Analysis**: computing (partial) radial distribution function histograms; calculating mean-square displacements; counting number of neighbour particles within given radius; calculating centre-of-mass motion in a trajectory; computing the velocity auto-correlation function; computing the Lindemann index for specific neighbour cutoff; calculating the bond angle between coordinated atoms and computing a histogram.

6.3 Modeling Fayalite

As reported in Section 6.1.1 the major constituent of the Upper Mantle is olivine, a solid solution of $\{Mg,Fe\}_2SiO_4$. In [184] experimental data on the structure of fayalite (Fe_2SiO_4) is presented, in particular the coordination of iron and oxygen with varying pressure and temperature in fayalite melts. Simulation data for forsterite (Mg_2SiO_4) from *ab-initio* molecular dynamics [118] and for a Mid-Ocean Ridge Basalt (MORB) composition from classical MD [88] is shown to be in good agreement but no MD data for fayalite is available. Also, at ambient pressure a dual-peak in the radial distribution function at 1.93Å and 2.18Å is observed [184], and attributed to 4-fold and 6-fold coordinated Fe respectively. To understand these features, computational modeling of liquid fayalite was carried out using CP2K.

6.3.1 Classical Force Fields

Three classical potential models were tested: from Pedone [168], a Morse-type potential with an explicit short-range repulsive term for improved high-temperature behaviour:

$$U_{ij}(r) = \frac{Z_i Z_j e^2}{r} + D_{ij}[\{1 - \exp(-a_{ij}(r - r_0)\}^2 - 1] + \frac{C_{ij}}{r^{12}}$$

From Guillot [87], a Buckingham-type potential:

$$U_{ij}(r) = \frac{Z_i Z_j e^2}{r} + B_{ij} \exp(-r_{ij}/\rho_{ij}) - C_{ij}/r^6$$

And from Walker [215], another Buckingham potential with an additional harmonic bond-angle potential constraining the O-Si-O tetrahedral bonds to an angle of 109.47°. The Walker model also employs a 'shell-model' [59] for oxygen, where instead of a single particle with charge 2−, the ions are modeled as a heavy core with charge 0.848+ connected to a light electron 'shell' with charge 2.848− via a harmonic spring, attempting to capture the effect of electronic polarisation. While the additional detail in the model might seem promising, the use of the bond-angle term limits the transferability of this force-field to situations where the structure of the SiO_4 tetrahedra are strictly maintained, and so it not useful for modeling melts where Si-O-Si bridging may occur, for example.

Guillot's and Pedone's models were found to give equilibrium lattice constants in reasonable agreement with experimental data (see Table 6.2), although both somewhat underestimate the length *b*.

	a (Å)		b (Å)		c (Å)		Volume (Å^3)	
Experiment[76]	4.82		10.48		6.09		305.59	
Pedone (this work)	4.89	+1.5%	10.27	-2.0%	6.08	+0.1%	301.41	+0.1%
Guillot (this work)	4.86	+0.9%	10.20	-2.6%	6.03	-1.0%	298.97	-2.2 %

Table 6.2 *Comparison of computed and experimental lattice parameters of fayalite.*

From the output of molecular dynamics calculations, radial (pair) distribution functions can be computed directly to give insight into the equilibrium structure. The all-atom RDF is defined as:

$$g(r) = 4\pi r^2 \rho dr$$

Where ρ is the number density i.e. the number of atoms per unit volume, so $g(r)$ is essentially the probability of finding an atom at spacing r from a reference atom, relative to an ideal gas. Corresponding pair (or partial) distribution functions between atoms of particular species can also be defined. In practice, the RDF was computed by calculating the distance between each atom pair, using the minimum-image convention, and creating a histogram of distances with bin size $\delta r = 0.005$Å. To avoid complications in the analysis relating to periodic boundary conditions, only distances up to 5Å (less than half the shortest cell dimension) are considered. The histogram is then normalised by

$$1/(4\pi n_{pairs} r^2 \delta r)$$

to obtain the RDF. The first coordination number (the number of atoms to be found within a given distance r of a chosen atom) can be computed by integrating $g(r)$ from $r = 0$ to the first minimum.

$$n = 4\pi \int_0^{r_{min}} r^2 g(r) \rho dr$$

The radial distribution functions of fayalite at ambient and high temperature are shown in Figures 6.4 and 6.5 respectively. The first peak in the crystalline structure corresponds to the SiO_4 complex which is correctly identified in 4-fold coordination with a bond length of 1.53 - 1.61Å, depending on the model. The second peak at 2.04 - 2.11Å is the 6-fold coordinated Fe-O octahedron.

In the melt (2250K), both Guillot's and Pedone's models show very little change in the mean Si-O distance, although the peaks are somewhat more widely spread indicating larger vibrations with the increased temperatures. The second peak is similarly smeared out, but remains 6-fold coordinated, and there is no indication of the 'shoulder' observed in [184].

6.3.2 DFT calculations

The same system was simulated using *ab-initio* MD calculations in the hope that this might better model the unknown physical behaviour responsible for the iron coordination splitting. However, due to the presence of iron, two extensions to standard Kohn-Sham DFT described in Section 2.1.1 are required.

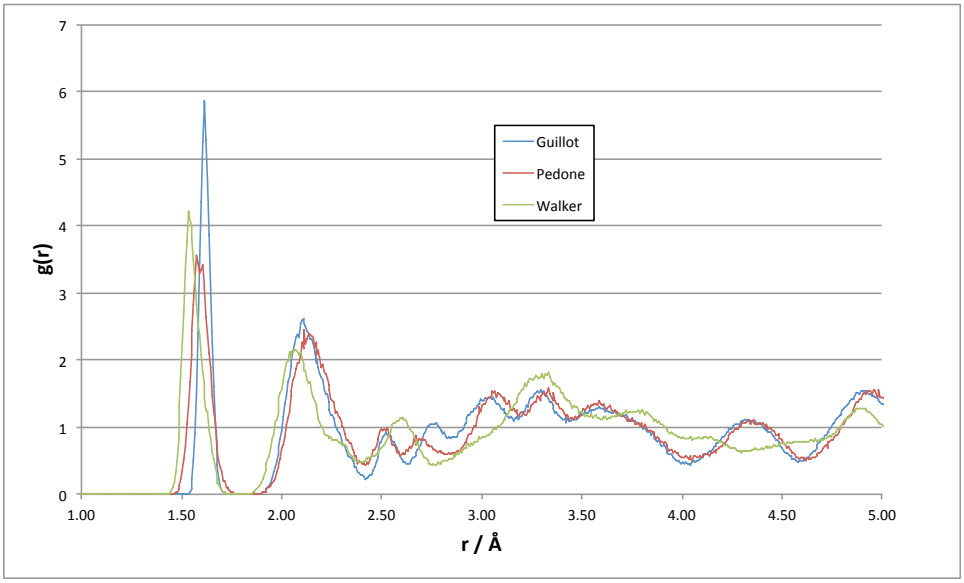


Figure 6.4 *Computed radial distribution functions of crystalline fayalite at 300K, 1 atm.*

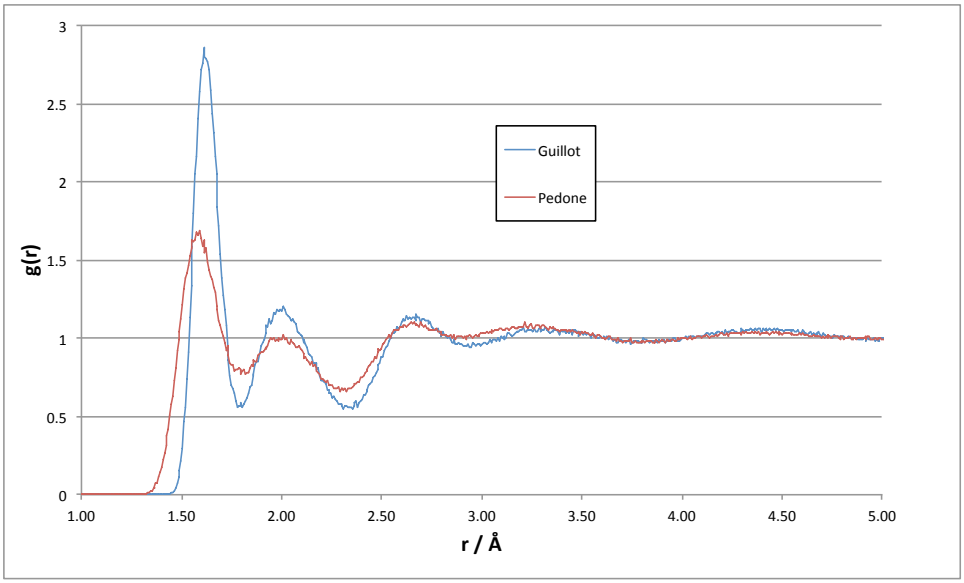


Figure 6.5 *Computed radial distribution functions of liquid fayalite at 2250K, 1 atm.*

It is a well-known failure of DFT (see e.g. [12]) that use of standard Local Density or Generalised Gradient Approximations for the Exchange-Correlation functional leads to prediction of a metallic ground state for transition metal oxides, due to spurious delocalisation of d electrons. The DFT+U scheme corrects for this by applying an empirical energy penalty (U) to these delocalised states, driving the electronic minimisation into the correctly localised insulating ground state. DFT+U has been applied with success to fayalite by Cococcioni *et al* [53, 54]. The implementation of this method in CP2K is based on the work of Dudarev [61, 62].

In addition, since iron is magnetic (the total spin is non-zero) it is no longer valid to ignore electron spin, allowing two 'identical' electrons to occupy each orbital. Instead of a single electronic density $n(\mathbf{r})$, two spin densities corresponding to spin up (α) and spin down (β) are computed and the total electron density is given by a sum:

$$n(\mathbf{r}) = n_\alpha(\mathbf{r}) + n_\beta(\mathbf{r})$$

The separation of spin into two classes (up and down), is itself an approximation since it neglects the fact that spin is a vector quantity. In particular for fayalite, the spins on some iron atoms are known to be non-collinear. The exact effect of this approximation is not known, but is assumed to be valid based on the results of [53], for example.

Attempting to run MD or geometry optimisation proved to be problematic. In particular, at some time steps the SCF procedure would fail to converge, at which point the calculation should be terminated to avoid jumps in the conserved quantity and resulting unphysical dynamics. The root cause appears to be in the calculation of the +U correction, given by:

$$E^U = \frac{1}{2} U_{eff} \sum_\mu (q_\mu - q_\mu^2)$$

where q_μ is the occupancy of a given state μ computing using Mulliken population analysis [158]. Since $q_\mu \leq 1$, the correction should always be positive. However, in some cases the population analysis returns states with occupancy > 1, resulting in a negative contribution to the total energy, causing problems for SCF convergence. Use of the Löwdin occupancy analysis model is recommended to overcome this,

but calculation of forces with this method is currently unimplemented in CP2K.

In addition, a number of parameters such as the plane-wave cutoff, choice of basis set, SCF convergence criteria and compensation for centre-of-mass drift had not yet been systematically studied, so further DFT computations for fayalite were abandoned at this stage.

6.4 Modeling Forsterite

To overcome some of these issues, a more comprehensive modeling study of forsterite (ceMg2SiO4) was carried out as a simpler analogue to fayalite in the sense that it is non-magnetic (so DFT calculations may ignore electron spin), and the lack of *d*-electrons means the DFT+U correction is likewise not required. The aims of this approach were to ensure that the basic computational methodology, tools and analysis scripts could be used to reliably simulation nesosilicates in both solid and melt using long-timescale MD, and also to compare and contrast the physical and structural properties obtained via classical and *ab initio* methods with available experimental reference data.

6.4.1 Simulation setup

All calculations are carried out using the CP2K program [105, 127]. An initial structure was set up using lattice parameters and ionic coordinates from [76], and then Molecular Dynamics was carried out in the isobaric isothermal (NPT) ensemble with a set temperature and external pressure of 1 atm. Thermostatting was achieved using a length-3 Nosé-Hoover chain [160, 161], with a time constant of 1ps. A timestep of 1.0fs and 0.5fs was used for the classical and *ab initio* calculations respectively. In each simulation an equilibriation phase at the start of the run of up to 15ps (depending on the temperature) was discarded and structures and averages were calculated from at least 10ps of *ab initio* or 120ps of classical MD.

The classical simulations use the force-field of Pedone *et al* [168], with a cut-off of 5.5Å. Long-range forces are computed using the Smoothed Particle-Mesh Ewald (SPME) summation [67]. A 3x3x3 supercell gives a total of 756 atoms.

Density Functional Theory (DFT) calculations are carried out within the Gaus-

sian and Plane Waves (GPW) dual-basis scheme [141, 210, 211], using Goedecker-Teter-Hutter (GTH) pseudopotentials [82, 95, 121] and the Perdew-Burke-Ernzerhof (PBE) [169] exchange-correlation functional within the Generalised Gradient Approximation. A double-zeta valence + polarization (DZVP) Gaussian basis set was used along with a 600 Ry cut-off for the planewave expansion of the electronic density. Due to the increased computational cost, a 2x1x2 supercell was used giving a total of 112 atoms.

6.4.2 Cell Parameters

Table 6.3 compares the lattice parameters in ambient conditions of temperature and pressure for forsterite from experimental studies and computer simulation. The two experimental results from Fujino *et al* [76], and Gillet *et al* [80], both using X-ray diffraction, are in good agreement. Data from classical molecular dynamics [168] is in reasonable agreement with CP2K simulation data, using the same potential model. The DFT calculations from [39] underestimate the cell volume, whereas the CP2K calculations overestimate the cell parameters.

	a (Å)		b (Å)		c (Å)		Volume (Å^3)	
Exp. 1[76]	4.75		10.19		5.98		289.58	
Exp. 2[80]	4.76	+0.2%	10.20	+0.1%	5.99	+0.2%	290.61	+0.3%
DFT (this work)	4.84	+1.9%	10.39	+2.0%	6.10	+2.0%	307.23	+6.1%
DFT (LDA, zero P) [39]	4.64	-2.3%	9.99	-2.0%	6.07	+1.5%	281.67	-2.7%
Classical (this work)	4.82	+1.4%	10.33	+1.4%	6.06	+1.3%	301.41	+4.1%
Classical [168]	4.84	+1.9%	10.19	-0.0%	6.00	+0.4%	296.24	+2.3%

Table 6.3 *Comparison of lattice parameters of forsterite at 300K, 1atm.*

6.4.3 Thermal Expansion

Using Molecular Dynamics at temperatures of 300K-3600K in steps of 300K, we compute the linear expansion coefficients $a/a_{300K}, b/b_{300K}, c/c_{300K}$ and volume expansion coefficient v/v_{300K} both below and above the melting point. The results from classical and *ab-initio* MD are compared with experimental data from Bouhifd *et al* [37] (see Figure 6.6). Following Bouhifd's notation, the expansion data (up to the melting point) is fitted to expressions of the form:

$$\ln(p_i(T)/p_i(T_0)) = \alpha_{i0}T + \alpha_{i1}/2T^2$$

where p_i are the lattice parameters, giving a thermal expansion coefficient:

$$\alpha_i \equiv \frac{1}{p_i}\left(\frac{\partial p_i}{\partial T}\right)_P = \alpha_{i0} + \alpha_{i1}T$$

Both the DFT and classical MD-derived data show significantly larger expansion than the experimental results. The DFT appears somewhat better in this regard. Also, both classical and DFT simulations show isotropic expansion, i.e., $\alpha_a = \alpha_b = \alpha_c$, compared with the expected larger expansions along the b and c axes. This is due to the isotropic barostatting scheme employed, and methods to overcome this are proposed in Section 6.5.2. The discontinuities in the expansion observed between 2100-2400K (classical) and 2400-2700K (DFT) indicate a sudden change in density and are correlated with the expected melting point of 2163K (further discussion in Section 6.4.4). The DFT expansion coefficient at 3600K is much higher than expected, both compared to the classical run and forward projection of the DFT data. This simulation was extended to 50 ps of dynamics, but the cell still showed large fluctuations, rather than relaxing to a clear equilibrium. This is mainly due to further expansion of the distance between Mg and O species. The SiO_4 tetrahedra still appear tightly bound at this temperature.

The calculated volume thermal expansion coefficient α is compared to Bouhifd's data as well as that of Gillet [80], Matsui [149] and Suzuki [200] in Figure 6.7. The trend of increasing expansion with temperature is consistent with experimental data but both the initial rate of expansion α_0 and the increase in expansion with the temperature α_1 are overestimated by both classical and DFT models.

6.4.4 Melting point

To try to accurately quantify whether a particular simulation has melted or not, two methods have been employed. Firstly, the mean-squared displacement of atoms can be calculated:

$$msd(t) = 1/N \sum_{i=1}^{N} (x_i(t) - x_i(0))^2$$

In a solid, atoms vibrate around their equilibrium site, and so the MSD should tend to a (small) constant in the limit of large t. In a liquid, atoms diffuse in a

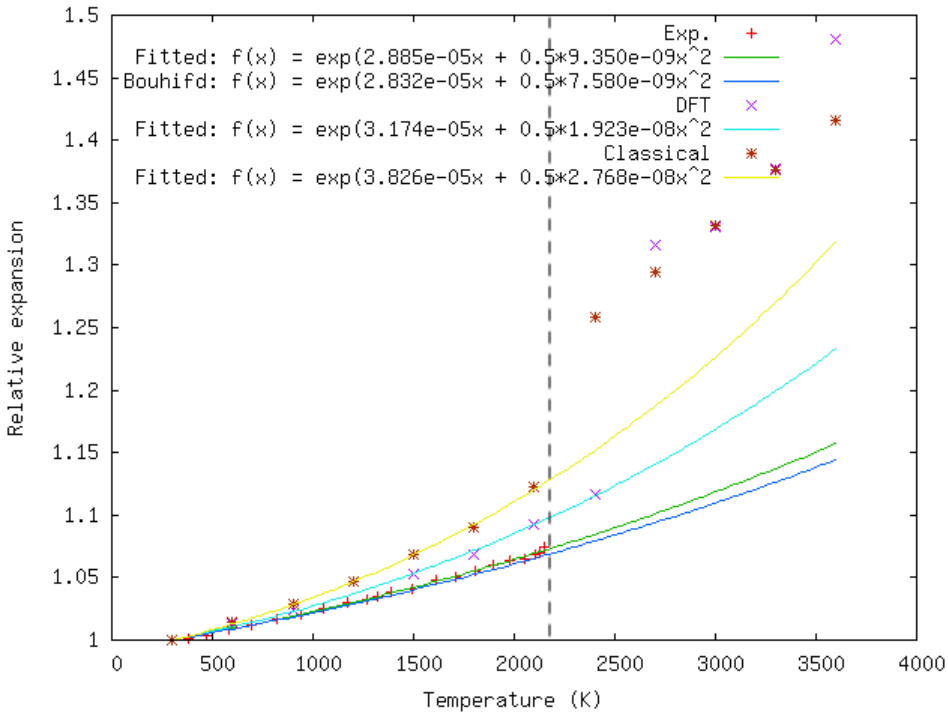


Figure 6.6 *Relative volume expansion of forsterite at ambient pressure comparing experimental data with DFT and classical simulations. The experimental melting point of 2163K is shown.*

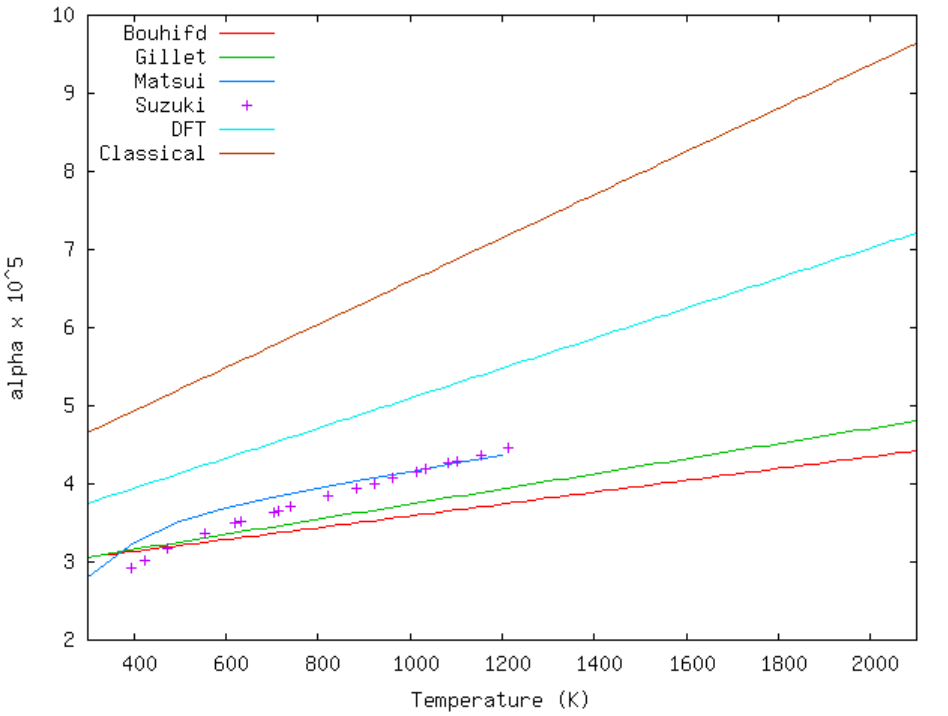


Figure 6.7 *Volume thermal expansion coefficient α of forsterite comparing experimental data with DFT and classical simulations up to the melting point.*

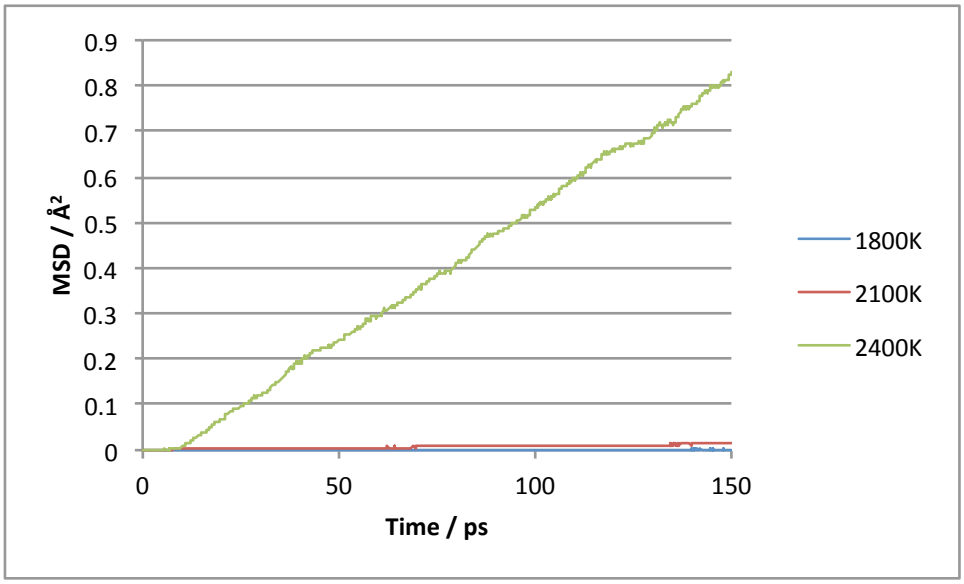


Figure 6.8 *Computed mean-squared displacements for classical MD simulation of forsterite above and below the melting point.*

random walk, and so the MSD is expected to grow linearly with time.

Figures 6.8 and 6.9 show the calculated MSDs for simulations above and below the known melting point of 2163K. In the classical case the simulations at 1800K and 2100K are clearly still solid, while the 2400K run exhibits typical melt behaviour after 5ps i.e. the melting point seems to be estimated correctly. In the DFT data, melting is not achieved until a temperature of 2700K. It is possible that the 2400K DFT simulation is in a meta-stable superheated state, and a simulation starting with a typical high-T configuration and cooling to 2400K might well result in a liquid state. A better approach to determining the melting point is discussed in Section 6.5.1.

Capturing the atomistic-scale detail of the melting process may also be investigated by defining a local Lindemann index [139]:

$$q_i = 1/n_{neigh} \sum_{j \in neigh} \frac{\sqrt{\langle r_{ij}^2 \rangle - \langle r_{ij} \rangle^2}}{\langle r_{ij} \rangle}$$

which gives a measure of the fluctuations over time in the separation between an atom and its nearest neighbours. In the case of the solid we expect this to by <1 and increasing with temperature, but not with time. In the melt, the Lindemann index should increase with time as atoms diffuse away from their initial neighbours. It should be possible to distinguish between Mg atoms, which

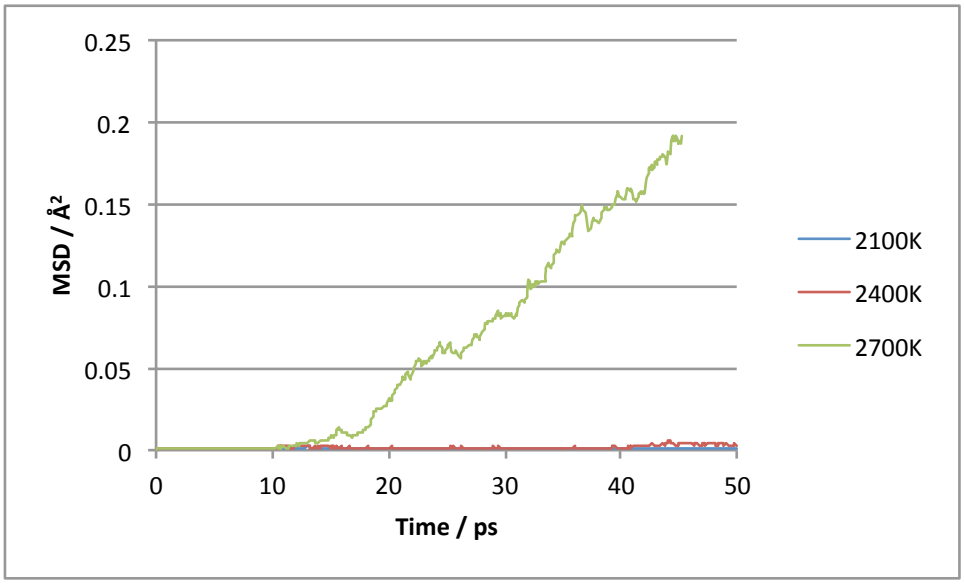


Figure 6.9 *Computed mean-squared displacements for* ab-initio *MD simulation of forsterite above and below the melting point.*

are expected to diffuse relative to the neighbouring O atoms, and the central Si atoms in the SiO_4 tetrahedra, which should have solid-like Lindemann indices. At the time of writing, this analysis has not been completed.

6.4.5 Heat capacity

From the MD calculations the Enthalpy can be computed since the Potential Energy U, applied pressure P and volume V are all known and thus the constant-pressure heat capacity:

$$H = U + PV$$

$$C_p = \left(\frac{\partial H}{\partial T}\right)_P$$

In Figure 6.10 the calculated values from classical and *ab-initio* MD are compared to experimental data (<300K [56], 300K-2100K [80]). At moderate temperatures these are in reasonable agreement with the experimental values. We did not simulate below 300K as neither model includes the nuclear quantum effects which are well-known (e.g. [212], Figure 1) to be be required to produce heat capacities which tend to zero at low temperature. Experimental heat capacities of forsterite melt are difficult to obtain, but are expected to be in the region of $225 - 295 JK^{-1}mol^{-1}$, according to [119], which is again consistent with the computed data.

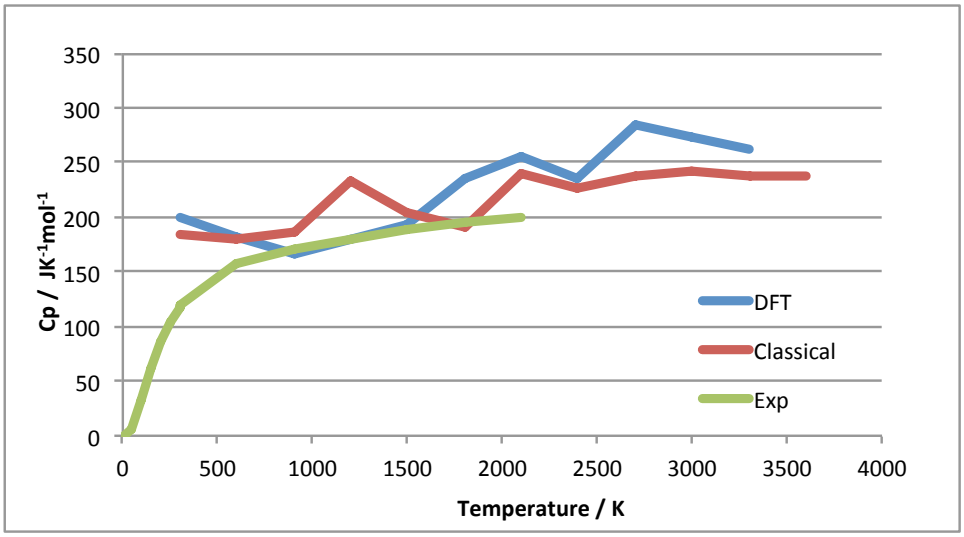


Figure 6.10 *Constant-pressure molar heat capacity of forsterite computed from classical and DFT MD, compared with experimental data.*

6.4.6 Structure

The radial distributions of crystalline (300K) and liquid (3000K) forsterite from both DFT and classical MD (discarding data from the equilibriation phase) were computed using the same technique as for fayalite and are compared in Figures 6.11 and 6.12. Representative structures from the solid and melt are shown in Figures 6.13 and 6.14. The first peak in the crystalline structure corresponds to the SiO_4 tetrahedron with an average bond length of 1.66Å (DFT) and 1.59Å (Classical) in 4-fold coordination. The second peak is the Mg−O complex, which is 6-fold coordinated and has interatomic distance 2.15Å (DFT) and 2.11Å (Classical). The classical RDF has two small peaks at 2.52Å and 2.67Å corresponding to the nearest neighbour O−O distances, and the next peak at 3.0Å is the average distance to the nearest Oxygen atom in the neighbouring tetrahedra. The DFT data gives slightly different peaks - the first O−O pairing has distance 2.60Å but this overlaps another broad peak with two tips at 2.84Å and 3.02Å so it is unclear whether these distances refer to neighbours within an SiO_4 tetrahedron or between neighbouring tetrahedra. The peak at 3.23Å (classical) and 3.26Å (DFT) is the nearest neighbour Mg−Mg pairing.

Comparing computed data with the measurements of Hazen [98] at 300K and 1 atm, an Si−O distance of 1.63Å is reported, intermediate between the computed values. The Mg−O distance is given as 2.11Å, and the O−O distances within an SiO_4 group are 2.5Å and 2.7Å i.e. the tetrahedron is distorted and some O

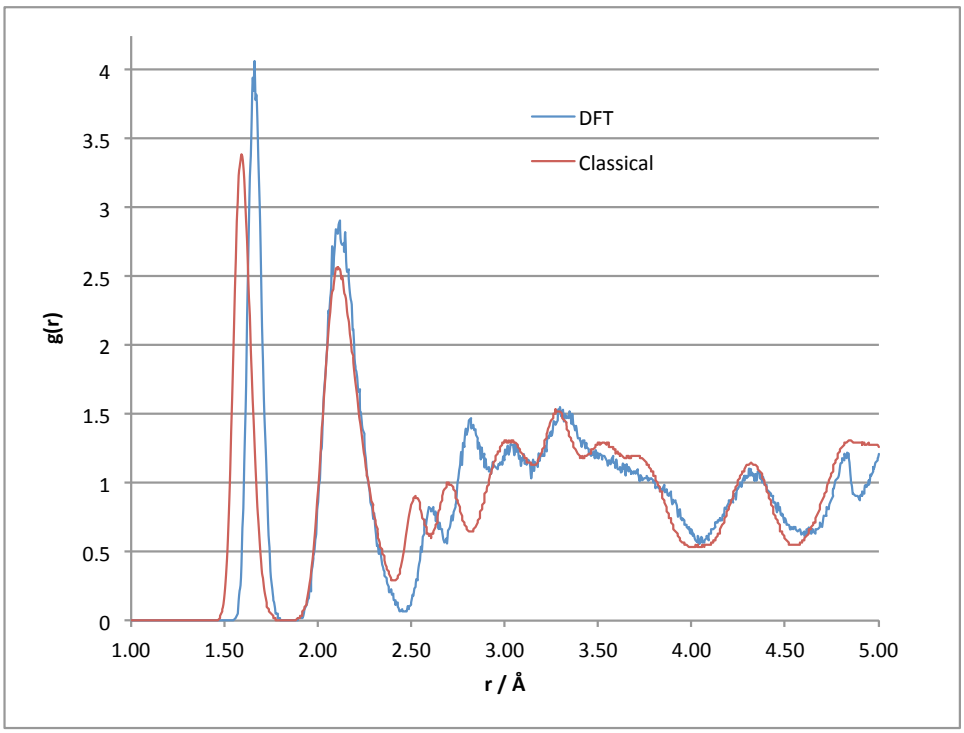


Figure 6.11 *Computed radial distribution functions of crystalline forsterite at 300K, 1 atm.*

pairs are slightly closer than others. The mean distance to the nearest neighbour outwith a tetrahedron is 2.98Å - these distances are in close agreement with the results from classical MD.

In the liquid forsterite, there are only three clearly identifiable peaks. The first is the Si−O tetrahedron, which has a maximum at around 1.63Å (DFT) and 1.58Å (Classical). While the peak is wider than that computed in the crystal - the bonds are vibrating at larger amplitude due to the increased kinetic energy in the system - the mean bond length is unchanged, which suggests the tetrahedra remain intact even at high temperature, and the expansion and liquid state are due to breakdown of the Mg−O octohedra. The intact (although distorted) tetrahedra are clearly visible in Figures 6.13b and 6.14b. The second peak, representing the nearest Mg−O neighbour, is very broad and peaks at 1.97Å (Classical) and 2.00Å (DFT). These are in fact smaller than the corresponding distances in the crystal, but the coordination number computed by integrating to the next minima in the Mg−O partial distribution function suggests a coordination of <6, and while this is somewhat unclear due to overlap with the next peak, may represent a loose tetrahedral arrangement with closer Mg−O spacing. The final (and very broad) peak is the O−O neighbours, which have mean separation of 2.7Å (DFT) and

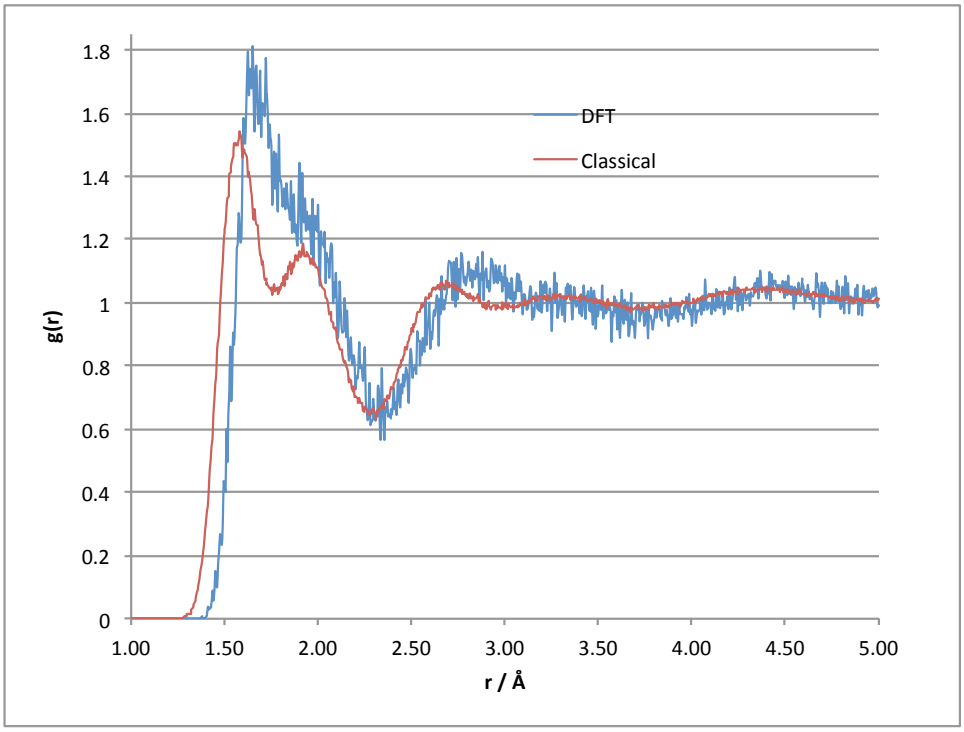


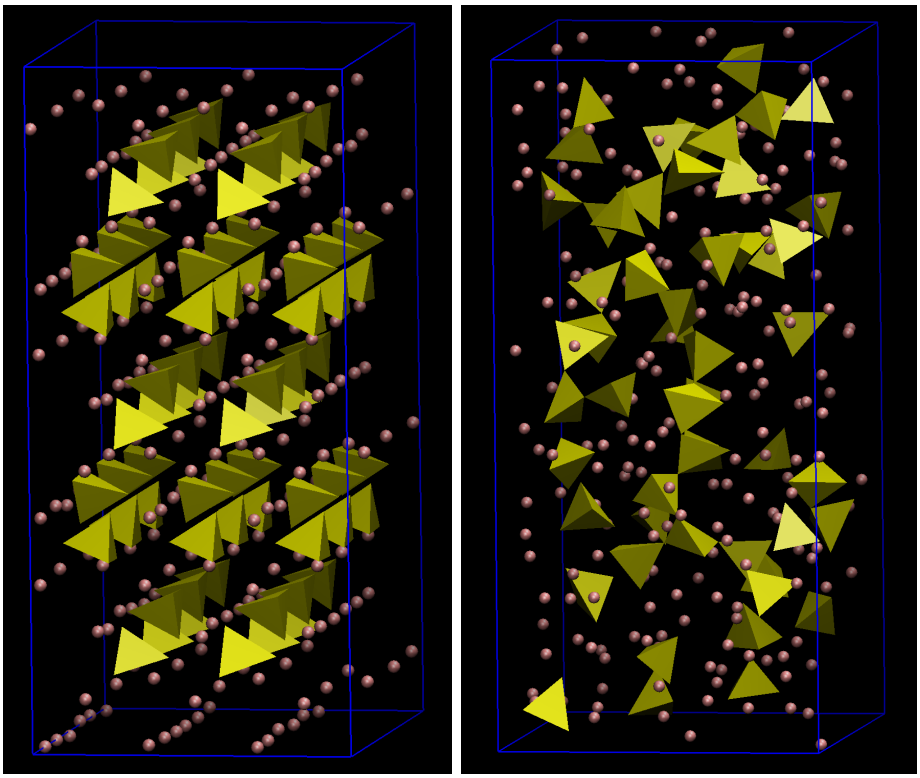
Figure 6.12 *Computed radial distribution functions of liquid forsterite at 3000K, 1 atm.*

2.67Å (Classical).

For comparison with radial distribution functions produced from scattering experiments, the partial distribution functions should be reweighted by a factor:

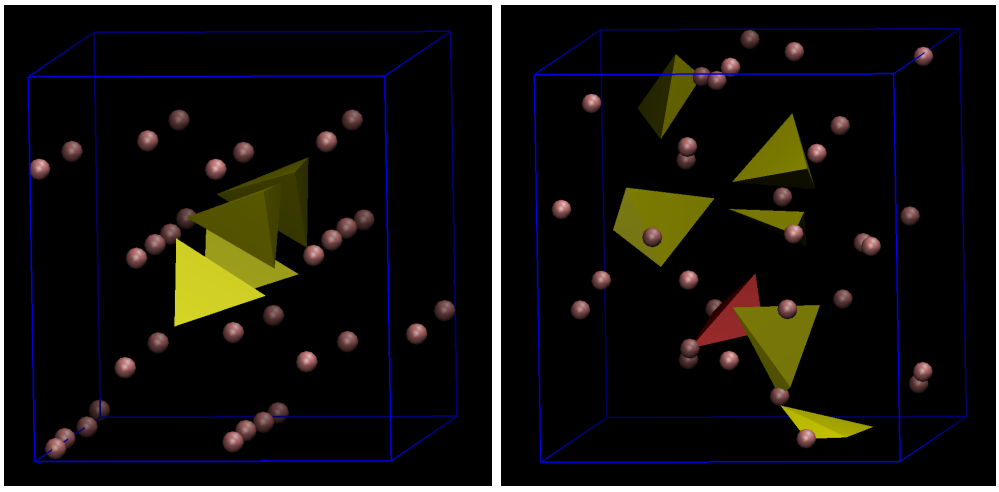
$$W_{\alpha\beta} = \frac{c_\alpha c_\beta f_\alpha f_\beta}{(\sum_\alpha c_\alpha f_\alpha)^2}$$

where c_α is the relative proportion of atoms of species α (i.e. 2/7 for Mg etc.) and f_α is the atomic number. This takes account of the relative strength of scattering from each species, although not the dependence of the scattering on frequency. To do so it would be better to calculate the partial structure factors directly from the MD trajectory data, reweight these at each frequency, and Fourier transform to obtain a radial distribution function which could be directly compared with experimental data.



(a) *300K* **(b)** *3000K*

Figure 6.13 *Representative structures of forsterite from classical simulations ($3 \times 3 \times 3$ supercell). Yellow tetrahedra represent SiO_4 groups and the pink spheres the Mg cations.*



(a) *300K* **(b)** *3000K*

Figure 6.14 *Representative structures of forsterite from DFT simulations ($2 \times 1 \times 2$ supercell). Yellow tetrahedra represent SiO_4 groups and the pink spheres the Mg cations.*

6.5 Summary

A small number of open questions (below) remain that would need to be resolved to complete testing of the forsterite models, but reasonably reliable models have been demonstrated and compare well to experiment. For fayalite, more fundamental issues remain - and significant effort would be required to implement both DFT+U forces with an accurate population analysis method and possible non-collinear spins in CP2K, or to use an alternative code such as Quantum Espresso [79].

It is interesting to note that at the time this work was carried out, developing bespoke codes for simulation setup and analysis was necessary (see Section 6.2.1). At time of submission a number of powerful packages such as the Atomic Simulation Environment [129] and MDAnalysis [107] are now available which provide much of the low-level functionality such as file handing, period boundaries, translations, rotations etc. and scripting on top of these would now be a much better choice.

6.5.1 Protocol for phase-coexistence MD

To determine the melting point of material in a particular model (either classical or DFT), instead of heating a solid or cooling a liquid, a better approach is to equilibriate a system at a fixed pressure which contains both solid and liquid phase components. The resulting measured temperature where the two phases are in equilibrium is by definition the modeled melting point of the material. To set up the system two separate simulations of solid and liquid phases are equilibriated in the NVT ensemble close to the expected experimental melting point. Constant volume is required so that the two (periodic) cells can be subsequently joined.

Joining the two systems together to continue MD requires care. There are no 'standard' methods available and uses of the approach in the literature tend to be short on details [151]. The best way to achieve this is unclear at this stage, but some initial attempts have been made. Firstly, the solid phase is sheared to present a high Miller-index face at the periodic boundary. In tests with a 10 by 10 by 10 supercell, the crystal has been sheared by 1 unit cell's distance to present the (10 0 0) face. The intention is that the interface with the liquid should have a relatively low adsorption energy, to avoid the situation where adding/removing particles to/from the surface has a high energy penalty, forcing the system into

an unphysical local energy minimum. In addition, in the process of shearing care must be taken to maintain the connectivity of SiO_4 tetrahedra, since these are not expected to disassociate until well beyond the melting point.

Secondly, the (sheared) solid and liquid systems can be brought together. This is done by placing the two simulation cells adjacent then translating the liquid particles until they are as close as physically reasonable (a minimum interatomic distance of 1.5Å was used). Finally, the particles at the interface must be allowed to relax to fill the void which still separates the two phases. Experimentation with geometry optimisation and MD to achieve this was not completed. The difficulty lies in the fact that while geometry optimisation fails to find the coordinated motion of particles to close the 'interface gap', when using MD the particles rapidly accelerate into the gap causing a sudden increase in temperature. It is hoped this can be overcome by running successive very short MD stages, resetting the temperature after each stage to quench the excess kinetic energy out of the system, or by applying an unphysical 'drag' force to damp out excessive acceleration. These schemes could easily be implemented in MIST, as well as other approaches such as capping unphysically large forces.

Once the system has been set up, a longer NPE (constant pressure and total energy) MD run will be carried out to equilibriate the temperature across both phases and thus identify the melting point.

6.5.2 Constant-pressure ensemble with anisotropic scaling

Running Molecular Dynamics in a constant-pressure ensemble implies that the simulation cell can change volume, and in general all nine components of the three lattice vectors may change in response to non-uniform external stress. CP2K uses the Martyna-Tuckerman-Klein (MTK) [148] equations of motion, which are based on the earlier (and more commonly implemented) scheme of Parrinello and Rahman [164]. In both cases the dynamics of the cell as modeled using a set of *extended system variables*, namely a set of positions and momenta relating to 'barostats' of some mass which couple changes in the cell to the difference between the externally applied pressure and the internal pressure estimator computed by a Virial expansion.

Both the Parrinello-Rahman and MTK schemes allow for two variants: isotropic cell scaling, where only uniform expansions or contractions of the cell are allowed,

a single barostat is employed, and the internal pressure is a scalar; and fully flexible cell, where there are nine barostats, each coupled to a different degree of freedom of the cell, and the internal pressure is evaluated as a 3x3 tensor quantity. In CP2K these are referred to as `NPT_I` and `NPT_F` respectively. The isotropic scheme is clearly inadequate to describe olivines, which are known to show anisotropic expansion (see Section 6.4.3). However, the fully flexible cell approach is problematic for liquid simulations since the lack of elastic restoring forces of a crystal allows the cell angles to oscillate wildly, complicating the analysis and potentially increasing finite size effects since atoms will be more likely to 'see' neighbouring images when the cell becomes very long and thin.

A new scheme is proposed which allows anisotropic cell expansion via scaling of each lattice vector independently, each of which is coupled to one of three barostats. This will maintain the orthorhombic cell shape, simplifying post-processing, and also be applicable directly in liquid and phase-coexistence calculations. Implementation of this scheme natively within CP2K has was not completed at time of writing. An interface between CP2K and MIST would make introducing the scheme much simpler, but is left for future work.

Chapter 7

Conclusion

In this thesis, I have described the architecture and implementation of MIST, the Molecular Integration Simulation Toolkit, a C++ library which provides an abstraction layer over common MD codes to enable rapid development of new MD integration algorithms. MIST is freely available under a BSD license from `https://bitbucket.org/extasy-project/mist`.

The current release of the library contains implementations of eleven different integrators, and is interfaced via a C or Fortran API to five MD codes: NAMD-Lite, GROMACS, Amber, LAMMPS and Forcite. MIST provides a portable platform for the development of novel integrators, which can be implemented once in MIST and used with any of the MD codes interfaced to MIST. Although the original motivation for MIST was to enable development of new integrators for biomolecular simulation (e.g. in Amber and GROMACS), MIST is also suitable for simulation of crystalline solids using LAMMPS.

I have demonstrated how MIST can be used in practice by implementing the Simulated Tempering scheme of Nguyen *et al* [159, 221] in combination with Langevin Dynamics using a 'BAOAB' splitting [132] and applying it to study the free energy landscape of Alanine-12 using GPU-accelerated GROMACS. MIST has also been used by collaborators to develop new tempering schemes such as Continuous Tempering [81] and Infinite Switch Simulated Tempering [146], proving that it is possible for other researchers to extend the library and achieve production-quality results.

In serial, parallel and GPU-accelerated configurations I have shown that in-

tegration using MIST introduces only a small overhead (typically 5-10%) compared to equivalent native calculations, with the exception of Amber's GPU implementation. In that case, the additional data transfer of the system state off the GPU introduces latency and synchronisation which slows the calculation down to performance comparable to GROMACS, where the native integration step is computed on the CPU and only forces are evaluated on the GPU. This performance overhead is argued to be an acceptable trade off for an expressive and easy-to-use interface for development of new algorithms and thus MIST provides an effective compromise between the two previous options: implementation in home-grown codes - rapid development, but low adoption, performance and tool support - and mainstream MD packages - slow development, but larger user base and higher performance.

I have also reported steps towards the development of reliable computational models of olivine melts based on both classical and *ab-initio* Molecular Dynamics. Comparisons between computed and experimental data for forsterite show that the simulation overpredicts by a few percent for lattice constants, thermal expansion coefficients and heat capacities, and melting points are estimated to within a few hundred degrees. Notwithstanding scaling, structural properties match well between experiment, classical and DFT data. The main reason for discrepancies in the thermal expansion coefficients is that the constant pressure integrator in CP2K allows for only isotropic expansion (or fully flexible cells - which are inappropriate for liquid calculations). Introducing an NPT scheme with only 3 cell degrees of freedom could be done rapidly via MIST, once an interface to CP2K is available. More accurate determination of the model melting points through phase-coexistence MD would also be made easier using the new integrators such as Langevin Dynamics in MIST.

7.1 Next Steps

As the initial phase of development of MIST described in this thesis concludes, there are several areas in which the library could be extended in future.

Firstly, from a software engineering point of view there are several areas for improvement. At present, error handling is done by return codes from the MIST API functions. These numeric codes are declared in the library interface but converting them to human-readable messages is done in the host code and

results in a lot of boilerplate code being added (which is duplicated across each supported code). A better approach would be to add an API call which allows the host code to obtain an error message for a given return code (similar to POSIX `strerror()`). This would reduce the amount of code which has to be added in to the host, and also make it possible to add further error codes without requiring all the host code patches to be updated. The internal error handling could also be improved by the use of C++ exceptions, which could be caught at the API layer and converted to error codes.

The lifecycle of the `Integrator` class could be improved by adding an `Init()` method where any one-off initialisation tasks have can be done, which would be called by MIST in between the integrator being constructed and the start of the first time-step. Currently, when the `Integrator` is constructed only the integrator parameters are available but not yet any of the system state variables so the complete initialisation has to be deferred to the first call to `Step()`, when all of the system state has been initialised. This could also be used to trigger any required reallocation when the number of particles per process changes after load balancing in a parallel run. This would result in reduced complexity in the `Step()` method, with the implementation looking even closer to the mathematical formulation of the algorithm than at present.

While integrator developers can benefit from OpenMP and GPU acceleration without code changes, they must still take care to include explicit MPI calls for cases like global reductions. Since one of the goals is to have an abstraction where developers don't need to be familiar with parallel programming, any required MPI should be wrapped in some convenience functions that are provided by MIST. Other similar improvements would be support for creating and managing extended state variables. The MPI parallelisation of the constraint solver is also necessary to offer the same level of MIST functionality for shared and distributed memory cases.

To support ongoing use of MIST, it is necessary to update the supported versions of GROMACS and Amber to the most recent published versions, as well as including support for additional popular codes such as CP2K, NAMD, DL_POLY [203], or GULP [77]. Eventually, as MIST gains wider usage, it is expected that MIST support may eventually be merged into these packages directly rather than having to rely on the patching process. This would significantly lift the sustainability burden of keeping patches up-to-date with changes in the supported codes by pushing responsibility onto the code maintainers as well as exposing

MIST to a wider user community.

Secondly, the core `System` abstraction in MIST could be extended. The first possibility would be to extend the representation of a particle from a point to spherical or ellipsoidal particles, essentially adding orientation and angular velocity state variable arrays as analogues to position, velocity and having accessors for moments of inertia and torques similarly to masses and forces. This would enable the implementation of algorithms such as [63].

Another useful extension would be to support multiple time-stepping schemes such as RESPA [206] and modern extensions [133]. The key change would be to allow extend both the time stepping and force updates with an integer parameter indicating which 'level' of the nested time steps to update. This would require support from the host code, of course, so might not be possible in all cases and would certainly be a radical change to the assumption that we can simply update forces with a single 'black-box' callback from MIST! To overcome the performance overhead of using MIST with Amber on GPU it is possible to envisage a hybrid/multiple time stepping scheme where MIST is used to integrate the outer timestep for slow degrees of freedom such as a thermostat, and the inner timestep for faster motions such as bond vibrations is integrated directly on the GPU using Amber's native integrator.

Bibliography

[1] "CPMD." http://www.cpmd.org/, . Copyright IBM Corp 1990-2008, Copyright MPI für Festkörperforschung Stuttgart 1997-2001.

[2] "List of quantum chemistry and solid-state physics software." http://en.wikipedia.org/wiki/List_of_quantum_chemistry_and_solid-state_physics_software, . Accessed: 20-07-2020.

[3] "All-Atom Empirical Potential for Molecular Modeling and Dynamics Studies of Proteins." *The Journal of Physical Chemistry B* 102, 18: (1998) 3586–3616. https://doi.org/10.1021/jp973084f. PMID: 24889800.

[4] Abraham, M. J., T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl. "GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers." *SoftwareX* 1–2: (2015) 19 – 25. https://doi.org/10.1016/j.softx.2015.06.001.

[5] Ackland, G. J. "Rapid Equilibration by algorithmic quenching the ringing mode in molecular dynamics." *MRS Advances* 1, 42: (2016) 2857–2865. https://doi.org/10.1557/adv.2016.382.

[6] Ackland, G.J. and D'Mellow, K. and Daraszewicz, S.L. and Hepburn, D.J. and Uhrin, M. and Stratford, K. "The MOLDY short-range molecular dynamics package." *Computer Physics Communications* 182, 12: (2011) 2587–2604. https://doi.org/10.1016/j.cpc.2011.07.014.

[7] Alfè, D., M. Gillan, and G. Price. "Composition and temperature of the Earth's core constrained by combining ab initio calculations and seismic data." *Earth and Planetary Science Letters* 195, 1–2: (2002) 91 – 98. https://doi.org/10.1016/S0012-821X(01)00568-4.

[8] Allen, M. P., and D. J. Tildesley. *Computer Simulation of Liquids*. New York, NY, USA: Clarendon Press, 1989. https://doi.org/10.1093/oso/9780198803195.001.0001.

[9] Andersen, H. C. "Molecular dynamics simulations at constant pressure and/or temperature." *The Journal of Chemical Physics* 72, 4: (1980) 2384–2393. https://doi.org/10.1063/1.439486.

[10] ———. "Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations." *Journal of Computational Physics* 52, 1: (1983) 24 – 34. https://doi.org/10.1016/0021-9991(83)90014-1.

[11] Anderson, D. L. *New Theory of the Earth*. Cambridge University Press, 2007. https://doi.org/10.1017/CBO9781139167291.

[12] Anisimov, V. I., J. Zaanen, and O. K. Andersen. "Band theory and Mott insulators: Hubbard U instead of Stoner I." *Physical Review B* 44, 3: (1991) 943–954. https://doi.org/10.1103/PhysRevB.44.943.

[13] Anzellini, S., A. Dewaele, M. Mezouar, P. Loubeyre, and G. Morard. "Melting of Iron at Earth's Inner Core Boundary Based on Fast X-ray Diffraction." *Science* 340, 6131: (2013) 464–466. https://doi.org/10.1126/science.1233514.

[14] Ashcroft, N. W., and N. D. Mermin. *Solid State Physics*. Holt Saunders, 1976.

[15] Bachelet, G. B., D. R. Hamann, and M. Schlüter. "Pseudopotentials that work: From H to Pu." *Phys. Rev. B* 26: (1982) 4199–4228. https://doi.org/10.1103/PhysRevB.26.4199.

[16] Balay, S., S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang. "PETSc Web page." https://www.mcs.anl.gov/petsc, 2019. https://www.mcs.anl.gov/petsc.

[17] ———. "PETSc Users Manual." Technical Report ANL-95/11 - Revision 3.13, Argonne National Laboratory, 2020. https://www.mcs.anl.gov/petsc.

[18] Balay, S., W. D. Gropp, L. C. McInnes, and B. F. Smith. "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries." In *Modern Software Tools in Scientific Computing*, edited by E. Arge, A. M. Bruaset, and H. P. Langtangen. Birkhäuser Press, 1997, 163–202. https://doi.org/10.1007/978-1-4612-1986-6_8.

[19] Barth, E., K. Kuczera, B. Leimkuhler, and R. D. Skeel. "Algorithms for constrained molecular dynamics." *Journal of Computational Chemistry* 16, 10: (1995) 1192–1209. https://doi.org/10.1002/jcc.540161003.

[20] Berendsen, H. J. C., J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak. "Molecular dynamics with coupling to an external bath." *The Journal of Chemical Physics* 81, 8: (1984) 3684–3690. https://doi.org/10.1063/1.448118.

[21] Bernardi, R. C., M. C. Melo, and K. Schulten. "Enhanced sampling techniques in molecular dynamics simulations of biological systems." *Biochimica et Biophysica Acta (BBA) - General Subjects* 1850, 5: (2015) 872 – 877. https://doi.org/10.1016/j.bbagen.2014.10.019. Recent developments of molecular dynamics.

[22] Bethune, I. "Improving the performance of CP2K on HECToR." Technical report, 2009. http://www.hector.ac.uk/cse/distributedcse/reports/cp2k/cp2k_final_report.pdf.

[23] ———. "Improving the performance of CP2K on multi-core systems." Technical report, 2010. http://www.hector.ac.uk/cse/distributedcse/reports/cp2k02/cp2k02_final_report.pdf.

[24] Bethune, I., R. Banisch, E. Breitmoser, A. B. Collis, G. Gibb, G. Gobbo, C. Matthews, G. J. Ackland, and B. J. Leimkuhler. "MIST: A simple and efficient molecular dynamics abstraction library for integrator development." *Computer Physics Communications* 236: (2019) 224 – 236. https://doi.org/10.1016/j.cpc.2018.10.006.

[25] Bethune, I., E. Breitmoser, A. B. K. Collis, G. Gobbo, and B. J. Leimkuhler. "Interfacing novel integrators and existing Molecular Dynamics codes with the MIST library." In *Producing High Performance and Sustainable Software for Molecular Simulation Workshop, Supercomputing 2015*. 2015. https://www.era.lib.ed.ac.uk/bitstream/handle/1842/16531/04-Bethune-Interfacing_integrators_MD_codes_MIST_library.pdf.

[26] Bethune, I., E. Breitmoser, G. Gobbo, C. Matthews, and B. J. Leimkuhler. "MIST: Molecular Integration Simulation Toolkit." In *Computational Molecular Science 2015*. 2015. https://ibethune.github.io/files/MIST-CMS2015.pdf.

[27] Bethune, I., E. Breitmoser, and B. J. Leimkuhler. "Molecular Integration Simulation Toolkit: Interfacing novel integrators with Molecular Dynamics codes." In *International Society of Quantum Biology and Pharmacology (ISQBP) President's Meeting 2016*. 2016. https://ibethune.github.io/files/MIST_ISQBP.pdf.

[28] Bethune, I., F. Reid, and A. Lazzaro. "CP2K Performance from Cray XT3 to XC30." In *Proceedings of the Cray User Group (CUG)*. 2014.

[29] Bethune, I., S. Wheeler, S. Genheden, and J. W. Essex. "Implementation of Dual Resolution Simulation Methodology in LAMMPS." Technical report, 2016. https://www.archer.ac.uk/documentation/white-papers/lammps-elba/lammps-ecse.pdf.

[30] BIOVIA. "BIOVIA Materials Studio Forcite Plus." http://accelrys.com/products/datasheets/forcite-plus.pdf, .

[31] Bjelkmar, P., P. Larsson, M. A. Cuendet, B. Hess, and E. Lindahl. "Implementation of the CHARMM Force Field in GROMACS: Analysis of Protein Stability Effects from Correction Maps, Virtual Interaction Sites, and Water Models." *Journal of Chemical Theory and Computation* 6, 2: (2010) 459–466. https://doi.org/10.1021/ct900549r. PMID: 26617301.

[32] Blacket, P., E. Bullard, and S. Runcorn. "A Symposium on Continental Drift." *Philosophical Transactions of the Royal Society A* 258. https://doi.org/10.1017/S0016756800050676.

[33] Blöchl, P. E. "Projector augmented-wave method." *Phys. Rev. B* 50: (1994) 17,953–17,979. https://doi.org/10.1103/PhysRevB.50.17953.

[34] Blum, V., R. Gehrke, F. Hanke, P. Havu, V. Havu, X. Ren, K. Reuter, and M. Scheffler. "Ab initio molecular simulations with numeric atom-centered orbitals." *Computer Physics Communications* 180, 11: (2009) 2175 – 2196. https://doi.org/10.1016/j.cpc.2009.06.022.

[35] Boehler, R. "MELTING TEMPERATURE OF THE EARTH'S MANTLE AND CORE: Earth's Thermal Structure." *Annual Review of Earth and Planetary Sciences* 24, 1: (1996) 15–40. https://doi.org/10.1146/annurev.earth.24.1.15.

[36] Born, M., and R. Oppenheimer. "Zur Quantentheorie der Molekeln." *Annalen der Physik* 389, 20: (1927) 457–484. https://doi.org/10.1002/andp.19273892002.

[37] Bouhifd, M., D. Andfault, G. Fiquet, and P. Richet. "Thermal expansion of forsterite up to the melting point." *Geophysical Research Letters* 23, 10: (1996) 1143–1146. https://doi.org/10.1029/96GL01118.

[38] Box, G. E. P., and M. E. Muller. "A Note on the Generation of Random Normal Deviates." *Ann. Math. Statist.* 29, 2: (1958) 610–611. https://doi.org/10.1214/aoms/1177706645.

[39] Brodholt, J., A. Patel, and K. Refson. "An ab initio study of the compressional behavior of forsterite." *American Mineralogist* 81: (1996) 257–260. https://doi.org/10.2138/am-1996-1-233.

[40] Brown, W. M., A. Kohlmeyer, S. J. Plimpton, and A. N. Tharrington. "Implementing molecular dynamics on hybrid high performance computers – Particle–particle particle-mesh." *Computer Physics Communications* 183, 3: (2012) 449 – 459. https://doi.org/10.1016/j.cpc.2011.10.012.

[41] Brown, W. M., P. Wang, S. J. Plimpton, and A. N. Tharrington. "Implementing molecular dynamics on hybrid high performance computers – short range forces." *Computer Physics Communications* 182, 4: (2011) 898 – 911. https://doi.org/10.1016/j.cpc.2010.12.021.

[42] Broyden, C. G. "A new double-rank minimization algorithm." *Notices of the American Mathematics Society* 16.

[43] Buckingham, R. A. "The Classical Equation of State of Gaseous Helium, Neon and Argon." *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* 168: (1938) 264–283. http://www.jstor.org/stable/97239.

[44] Bullen, K., and B. Bolt. *An Introduction to the Theory of Seismology*. Cambridge University Press, 1947. https://doi.org/10.1111/j.1365-246X.1986.tb01083.x.

[45] Burns, R. G. *Mineralogical Applications of Crystal Field Theory*. Cambridge University Press, 1993. https://doi.org/10.1017/CBO9780511524899.

[46] Bush, I., and I. Todorov. "Optimisation of the Input and Output (I/O) in DL_POLY_3." Technical report, 2009. http://www.hector.ac.uk/cse/distributedcse/reports/DL_POLY01/DL_POLY01.pdf.

[47] Car, R., and M. Parrinello. "Unified Approach for Molecular Dynamics and Density-Functional Theory." *Phys. Rev. Lett.* 55: (1985) 2471–2474. https://doi.org/10.1103/PhysRevLett.55.2471.

[48] Case, D. A., T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods. "The Amber biomolecular simulation programs." *Journal of Computational Chemistry* 26, 16: (2005) 1668–1688. https://doi.org/10.1002/jcc.20290.

[49] Ceriotti, M., G. Bussi, and M. Parrinello. "Colored-Noise Thermostats à la Carte." *Journal of Chemical Theory and Computation* 6, 4: (2010) 1170–1180. https://doi.org/10.1021/ct900563s.

[50] Chadwick, F. M., N. H. Rees, A. S. Weller, T. Krämer, M. Iannuzzi, and S. A. Macgregor. "A Rhodium–Pentane Sigma-Alkane Complex: Characterization in the Solid State by Experimental and Computational Techniques." *Angewandte Chemie International Edition* 55, 11: (2016) 3677–3681. https://doi.org/10.1002/anie.201511269.

[51] Chen, M., M. A. Cuendet, and M. E. Tuckerman. "Heating and flooding: A unified approach for rapid generation of free energy surfaces." *The Journal of Chemical Physics* 137, 2: (2012) 024,102. https://doi.org/10.1063/1.4733389.

[52] Clark, S., M. Segall, and C. Pickard. "First principles methods using CASTEP." *Zeitschrift für Kristallographie* 220: (2005) 567–570. https://doi.org/10.1524/zkri.220.5.567.65075.

[53] Cococcioni, M., A. Dal Corso, and S. de Gironcoli. "Structural, electronic, and magnetic properties of Fe2SiO4 fayalite: Comparison of LDA and GGA results." *Physical Review B* 67, 9: (2003) 094,106. https://doi.org/10.1103/PhysRevB.67.094106.

[54] Cococcioni, M., and S. de Gironcoli. "Linear response approach to the calculation of the effective interaction parameters in the LDA+U method." *Physical Review B* 71, 3: (2005) 035,105. https://doi.org/10.1103/PhysRevB.71.035105.

[55] Cornell, W. D., P. Cieplak, C. I. Bayly, I. R. Gould, K. M. Merz, D. M. Ferguson, D. C. Spellmeyer, T. Fox, J. W. Caldwell, and P. A. Kollman. "A Second Generation Force Field for the Simulation of Proteins, Nucleic Acids, and Organic Molecules." *Journal of the American Chemical Society* 117, 19: (1995) 5179–5197. https://doi.org/10.1021/ja00124a002.

[56] Dachs, E., C. A. Geiger, V. von Seckendorff, and M. Grodzicki. "A low-temperature calorimetric study of synthetic (forsterite+fayalite) {(Mg2SiO4+Fe2SiO4)} solid solutions: An analysis of vibrational, magnetic, and electronic contributions to the molar heat capacity and entropy of mixing." *The Journal of Chemical Thermodynamics* 39, 6: (2007) 906–933. https://doi.org/10.1016/j.jct.2006.11.009.

[57] Del Ben, M., J. Hutter, and J. VandeVondele. "Second-Order Moller–Plesset Perturbation Theory in the Condensed Phase: An Efficient and Massively Parallel Gaussian and Plane Waves Approach." *Journal of Chemical Theory and Computation* 8: (2012) 4177–4188. https://doi.org/10.1021/ct300531w.

[58] Del Ben, M., O. Schütt, T. Wentz, P. Messmer, J. Hutter, and J. VandeVondele. "Enabling simulation at the fifth rung of DFT: Large scale RPA calculations with excellent time to solution." *Computer Physics Communications* 187: (2015) 120 – 129. https://doi.org/10.1016/j.cpc.2014.10.021.

[59] Dick, B. G., and A. W. Overhauser. "Theory of the Dielectric Constants of Alkali Halide Crystals." *Phys. Rev.* 112: (1958) 90–103. https://doi.org/10.1103/PhysRev.112.90.

[60] Duan, Y., C. Wu, S. Chowdhury, M. C. Lee, G. Xiong, W. Zhang, R. Yang, P. Cieplak, R. Luo, T. Lee, J. Caldwell, J. Wang, and P. Kollman. "A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations." *Journal of Computational Chemistry* 24, 16: (2003) 1999–2012. https://doi.org/10.1002/jcc.10349.

[61] Dudarev, S. L., D. N. Manh, and A. P. Sutton. "Effect of Mott-Hubbard correlations on the electronic structure and structural stability of uranium dioxide." *Philosophical Magazine Part B* 75, 5: (1997) 613–628. https://doi.org/10.1080/13642819708202343.

[62] Dudarev, S., G. Botton, and S. Savrasov. "Electron-energy-loss spectra and the structural stability of nickel oxide: An LSDA+ U study." *Physical Review B* 57, 3: (1998) 1505–1509. https://doi.org/10.1103/PhysRevB.57.1505.

[63] Dullweber, A., B. Leimkuhler, and R. McLachlan. "Symplectic splitting methods for rigid body molecular dynamics." *The Journal of Chemical Physics* 107, 15: (1997) 5840–5851. https://doi.org/10.1063/1.474310.

[64] Dziewonski, A., and D. Anderson. "Preliminary Reference Earth Model." *Physics of Earth and Planetary Interiors* 25: (1981) 297–356. https://doi.org/10.1016/0031-9201(81)90046-7.

[65] Eastman, P., J. Swails, J. D. Chodera, R. T. McGibbon, Y. Zhao, K. A. Beauchamp, L.-P. Wang, A. C. Simmonett, M. P. Harrigan, C. D. Stern, R. P. Wiewiora, B. R. Brooks, and V. S. Pande. "OpenMM 7: Rapid development of high performance algorithms for molecular dynamics." *PLOS Computational Biology* 13, 7: (2017) 1–17. https://doi.org/10.1371/journal.pcbi.1005659.

[66] Elber, R. "Perspective: Computer simulations of long time dynamics." *The Journal of Chemical Physics* 144, 6: (2016) 060,901. https://doi.org/10.1063/1.4940794.

[67] Essmann, U., L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen. "A smooth particle mesh Ewald method." *The Journal of Chemical Physics* 103: (1995) 8577–8593. https://doi.org/10.1063/1.470117.

[68] Ewald, P. P. "Die Berechnung optischer und elektrostatischer Gitterpotentiale." *Annalen der Physik* 369, 3: (1921) 253–287. https://doi.org/10.1002/andp.19213690304.

[69] Fass, J., D. Sivak, G. E. Crooks, K. A. Beauchamp, B. Leimkuhler, and J. Chodera. "Quantifying configuration-sampling error in Langevin simulations of complex molecular systems." *bioRxiv* https://doi.org/10.1101/266619.

[70] Feynman, R. P. "Forces in Molecules." *Phys. Rev.* 56: (1939) 340–343. https://doi.org/10.1103/PhysRev.56.340.

[71] Fletcher, R. "A new approach to variable metric methods." *The Computer Journal* 13. https://doi.org/10.1093/comjnl/13.3.317.

[72] Foiles, S. M., M. I. Baskes, and M. S. Daw. "Embedded-atom-method functions for the fcc metals Cu, Ag, Au, Ni, Pd, Pt, and their alloys." *Phys. Rev. B* 33: (1986) 7983–7991. https://doi.org/10.1103/PhysRevB.33.7983.

[73] Fowler, C. *The Solid Earth: An Introduction to Global Geophysics.* Cambridge University Press, 1990.

[74] Freddolino, P., A. Arkhipov, S. Larson, A. McPherson, and K. Schulten. "Molecular dynamics simulations of the complete satellite tobacco mosaic

virus." *Structure with Folding and design* 14, 3: (2006) 437–449. https://doi.org/10.1016/j.str.2005.11.014.

[75] Frenkel, D., and B. Smit. *Understanding Molecular Simulation*. Orlando, FL, USA: Academic Press, Inc., 2001, 2nd edition. https://doi.org/10.1016/B978-0-12-267351-1.X5000-7.

[76] Fujino, K., S. Sasaki, Y. Takeuchi, and R. Sadanaga. "X-ray Determination of Electron Distributions in Forsterite, Fayalite and Tephroite." *Acta Crystallographica* B37, 3. https://doi.org/10.1107/S0567740881003506.

[77] Gale, J. D. "GULP: A computer program for the symmetry-adapted simulation of solids." *J. Chem. Soc., Faraday Trans.* 93: (1997) 629–637. https://doi.org/10.1039/A606455H.

[78] Genovese, L., A. Neelov, S. Goedecker, T. Deutsch, S. A. Ghasemi, A. Willand, D. Caliste, O. Zilberberg, M. Rayson, A. Bergman, and R. Schneider. "Daubechies wavelets as a basis set for density functional pseudopotential calculations." *The Journal of Chemical Physics* 129, 1: (2008) 014,109. https://doi.org/10.1063/1.2949547.

[79] Giannozzi, P., S. Baroni, N. Bonini, M. Calandra, R. Car, C. Cavazzoni, D. Ceresoli, G. L. Chiarotti, M. Cococcioni, I. Dabo, A. Dal Corso, S. de Gironcoli, S. Fabris, G. Fratesi, R. Gebauer, U. Gerstmann, C. Gougoussis, A. Kokalj, M. Lazzeri, L. Martin-Samos, N. Marzari, F. Mauri, R. Mazzarello, S. Paolini, A. Pasquarello, L. Paulatto, C. Sbraccia, S. Scandolo, G. Sclauzero, A. P. Seitsonen, A. Smogunov, P. Umari, and R. M. Wentzcovitch. "QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials." *Journal of Physics: Condensed Matter* 21, 39: (2009) 395,502 (19pp). http://www.quantum-espresso.org.

[80] Gillet, P., P. Richet, F. Guyot, and G. Fiquet. "High-Temperature Thermodynamic Properties of Forsterite." *Journal of Geophysical Research* 96, B7: (1991) 11,805–11,816. https://doi.org/10.1029/91JB00680.

[81] Gobbo, G., and B. J. Leimkuhler. "Extended Hamiltonian approach to continuous tempering." *Phys. Rev. E* 91: (2015) 061,301. https://doi.org/10.1103/PhysRevE.91.061301.

[82] Goedecker, S., M. Teter, and J. Hutter. "Separable dual-space Gaussian pseudopotentials." *Physical Review B* 54, 3. https://doi.org/10.1103/PhysRevB.54.1703.

[83] Goldfarb, D. "A family of variable metric methdos derived by variational means." *Mathematics of Computation* 24. https://doi.org/10.2307/2004873.

[84] Gu, X., and C. Zhao. "Thermal conductivity of single-layer MoS2(1-x)Se2x alloys from molecular dynamics simulations with a machine-learning-based interatomic potential." *Computational Materials Science* 165: (2019) 74 – 81. https://doi.org/10.1016/j.commatsci.2019.04.025.

[85] Guennebaud, G., B. Jacob, et al. "Eigen v3." http://eigen.tuxfamily.org, 2010.

[86] Guidon, M., J. Hutter, and J. VandeVondele. "Robust Periodic Hartree-Fock Exchange for Large-Scale Simulations Using Gaussian Basis Sets." *Journal of Chemical Theory and Computation - J CHEM THEORY COMPUT* 5. https://doi.org/10.1021/ct900494g.

[87] Guillot, B., and N. Sator. "A computer simulation study of natural silicate melts. Part I: Low pressure properties." *Geochimica et Cosmochimica Acta* 71, 5: (2007) 1249–1265. https://doi.org/10.1016/j.gca.2006.11.015.

[88] ———. "A computer simulation study of natural silicate melts. Part II: High pressure properties." *Geochimica et Cosmochimica Acta* 71, 18: (2007) 4538–4556. https://doi.org/10.1016/j.gca.2007.05.029.

[89] Gullingsrud, J. "MINDY - A 'minimal' molecular dynamics program." https://www.ks.uiuc.edu/Development/MDTools/mindy/, University of Illinois at Urbana-Champaign, .

[90] Götz, A., M. Williamson, D. Xu, D. Poole, S. Grand, and R. Walker. "Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 1. Generalized Born." *Journal of chemical theory and computation* 8: (2012) 1542–1555. https://doi.org/10.1021/ct200909j.

[91] Hamelberg, D., J. Mongan, and J. A. McCammon. "Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules." *The Journal of Chemical Physics* 120, 24: (2004) 11,919–11,929. https://doi.org/10.1063/1.1755656.

[92] Hansmann, U. H. "Parallel tempering algorithm for conformational studies of biological molecules." *Chemical Physics Letters* 281, 1: (1997) 140 – 150. https://doi.org/10.1016/S0009-2614(97)01198-6.

[93] Hardy, D. J. "NAMD-Lite." http://www.ks.uiuc.edu/Development/MDTools/namdlite/, University of Illinois at Urbana-Champaign, 2007.

[94] Hartree, D. R., and W. Hartree. "Self-Consistent Field, with Exchange, for Beryllium." *Proceedings of the Royal Society of London. Series A - Mathematical and Physical Sciences* 150, 869: (1935) 9–33. https://doi.org/10.1098/rspa.1935.0085.

[95] Hartwigsen, C., S. Goedecker, and J. Hutter. "Relativistic separable dual-space Gaussian pseudopotentials from H to Rn C." *Physical Review B* 58, 7. https://doi.org/10.1103/PhysRevB.58.3641.

[96] Hashemian, B., D. Millán, and M. Arroyo. "Modeling and enhanced sampling of molecular systems with smooth and nonlinear data-driven collective variables." *The Journal of chemical physics* 139: (2013) 214,101. https://doi.org/10.1063/1.4830403.

[97] Hazel, A., C. Chipot, and J. C. Gumbart. "Thermodynamics of Deca-alanine Folding in Water." *Journal of Chemical Theory and Computation* 10, 7: (2014) 2836–2844. https://doi.org/10.1021/ct5002076. PMID: 25061447.

[98] Hazen, R. "Effects of temperature and pressure on the crystal structure of forsterite." *Am Mineral* 61, l: (1976) 1280–1293. http://www.minsocam.org/ammin/AM61/AM61_1280.pdf.

[99] Hehre, W. J., W. A. Lathan, R. Ditchfield, M. D. Newton, and J. A. Pople. "Gaussian 70." Quantum Chemistry Program Exchange, Program No. 237, .

[100] Hess, B., H. Bekker, H. J. C. Berendsen, and J. G. E. M. Fraaije. "LINCS: A linear constraint solver for molecular simulations." *Journal of Computational Chemistry* 18, 12: (1997) 1463–1472. https://doi.org/10.1002/(SICI)1096-987X(199709)18:12<1463::AID-JCC4>3.0.CO;2-H.

[101] Hohenberg, P., and W. Kohn. "Inhomogeneous Electron Gas." *Phys. Rev.* 136: (1964) B864–B871. https://doi.org/10.1103/PhysRev.136.B864.

[102] Hoover, W. G. "Canonical dynamics: Equilibrium phase-space distributions." *Phys. Rev. A* 31: (1985) 1695–1697. https://doi.org/10.1103/PhysRevA.31.1695.

[103] Hourahine, B., B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-z. Yu, and T. Frauenheim. "DFTB+, a software package for efficient approximate density functional theory based atomistic simulations." *The Journal of Chemical Physics* 152, 12: (2020) 124,101. https://doi.org/10.1063/1.5143190.

[104] Hruska, E., V. Balasubramanian, J. R. Ossyra, S. Jha, and C. Clementi. "Extensible and Scalable Adaptive Sampling on Supercomputers.", 2019. https://arxiv.org/abs/1907.06954.

[105] Hutter, J., M. Iannuzzi, F. Schiffmann, and J. VandeVondele. "CP2K: Atomistic Simulations of Condensed Matter Systems." *Wiley Interdisciplinary Reviews: Computational Molecular Science* 4, 1: (2014) 15–25. https://doi.org/10.1002/wcms.1159.

[106] Hutton, J. "Theory of the Earth; or an Investigation of the Laws observable in the Composition, Dissolution, and Restoration of Land upon the Globe." *Transactions of the Royal Society of Edinburgh* 1.

[107] Richard J. Gowers, Max Linke, Jonathan Barnoud, Tyler J. E. Reddy, Manuel N. Melo, Sean L. Seyler, Jan Domański, David L. Dotson, Sébastien Buchoux, Ian M. Kenney, and Oliver Beckstein. "MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations." In *Proceedings of the 15th Python in Science Conference*, edited by Sebastian Benthall, and Scott Rostrup. 2016, 98 – 105. https://doi.org/10.25080/Majora-629e541a-00e.

[108] Jackson, W. E., J. M. de Leon, G. E. Brown, G. a. Waychunas, S. D. Conradson, and J. M. Combes. "High-Temperature XAS Study of Fe2SiO4 Liquid: Reduced Coordination of Ferrous Iron." *Science (New York, N.Y.)* 262, 5131: (1993) 229–33. https://doi.org/10.1126/science.262.5131.229.

[109] Jing, Z., C. Liu, S. Y. Cheng, R. Qi, B. D. Walker, J.-P. Piquemal, and P. Ren. "Polarizable Force Fields for Biomolecular Simulations: Recent Advances and Applications." *Annual review of biophysics* 48. https://doi.org/10.1146/annurev-biophys-070317-033349.

[110] Jones, A., and B. Leimkuhler. "Adaptive stochastic methods for sampling driven molecular systems." *The Journal of Chemical Physics* 135, 8: (2011) 084,125. https://doi.org/10.1063/1.3626941.

[111] Jones, A., J. Crain, F. Cipcigan, V. Sokhan, M. Modani, and G. Martyna. "Electronically coarse-grained molecular dynamics using quantum Drude oscillators." *Molecular Physics* 111, 22-23: (2013) 3465–3477. https://doi.org/10.1080/00268976.2013.843032.

[112] Jones, J. E. "On the Determination of Molecular Fields. II. From the Equation of State of a Gas." *Proceedings of the Royal Society of London. Series A* 106, 738: (1924) 463–477. https://doi.org/10.1098/rspa.1924.0082.

[113] Jones, M. D., J. P. White, M. Innus, R. L. DeLeon, N. Simakov, J. T. Palmer, S. M. Gallo, T. R. Furlani, M. T. Showerman, R. Brunner, A. Kot, G. H. Bauer, B. M. Bode, J. Enos, and W. T. Kramer. "Workload Analysis of Blue Waters." *ArXiv* abs/1703.00924. https://arxiv.org/abs/1703.00924.

[114] Kapil, V., M. Rossi, O. Marsalek, R. Petraglia, Y. Litman, T. Spura, B. Cheng, A. Cuzzocrea, R. H. Meißner, D. M. Wilkins, B. A. Helfrecht, P. Juda, S. P. Bienvenue, W. Fang, J. Kessler, I. Poltavsky, S. Vandenbrande, J. Wieme, C. Corminboeuf, T. D. Kühne, D. E. Manolopoulos, T. E. Markland, J. O. Richardson, A. Tkatchenko, G. A. Tribello, V. Van Speybroeck, and M. Ceriotti. "i-PI 2.0: A universal

force engine for advanced molecular simulations." *Computer Physics Communications* 236: (2019) 214 – 223. https://doi.org/10.1016/j.cpc.2018.09.020.

[115] Kennet, B., and E. Engdahl. "Travel times for global earthquake location and phase association." *Geophysical Journal International* 105. https://doi.org/10.1111/j.1365-246X.1991.tb06724.x.

[116] Kennet, B., E. Engdahl, and R. Buland. "Constraints on seismic velocities in the earth from travel times." *Geophysical Journal International* 122. https://doi.org/10.1111/j.1365-246X.1995.tb03540.x.

[117] Kohn, W., and L. J. Sham. "Self-Consistent Equations Including Exchange and Correlation Effects." *Phys. Rev.* 140: (1965) A1133–A1138. https://doi.org/10.1103/PhysRev.140.A1133.

[118] de Koker, N., B. B. Karki, and L. Stixrude. "Thermodynamics of the MgO–SiO2 liquid system in Earth's lowermost mantle from first principles." *Earth and Planetary Science Letters* 361: (2013) 58–63. https://doi.org/10.1016/j.epsl.2012.11.026.

[119] de Koker, N. P., L. Stixrude, and B. B. Karki. "Thermodynamics, structure, dynamics, and freezing of Mg2SiO4 liquid at high pressure." *Geochimica et Cosmochimica Acta* 72, 5: (2008) 1427–1441. https://doi.org/10.1016/j.gca.2007.12.019.

[120] Kol, A., B. B. Laird, and B. J. Leimkuhler. "A symplectic method for rigid-body molecular simulation." *The Journal of Chemical Physics* 107, 7: (1997) 2580–2588. https://doi.org/10.1063/1.474596.

[121] Krack, M. "Pseudopotentials for H to Kr optimized for gradient-corrected exchange-correlation functionals." *Theoretical Chemistry Accounts* 114, 1-3: (2005) 145–152. https://doi.org/10.1007/s00214-005-0655-y.

[122] Kresse, G., and J. Hafner. "Ab initio molecular dynamics for liquid metals." *Phys. Rev. B* 47: (1993) 558–561. https://doi.org/10.1103/PhysRevB.47.558.

[123] Kumar, S., J. M. Rosenberg, D. Bouzida, R. H. Swendsen, and P. A. Kollman. "THE weighted histogram analysis method for free-energy calculations on biomolecules. I. The method." *Journal of Computational Chemistry* 13, 8: (1992) 1011–1021. https://doi.org/10.1002/jcc.540130812.

[124] Kustowski, B., G. Ekström, and A. M. Dziewoński. "Anisotropic shear-wave velocity structure of the Earth's mantle: A global model." *Journal of Geophysical Research: Solid Earth* 113, B6. https://doi.org/10.1029/2007JB005169.

[125] Kutta, M. W. "Beitrag zur näherungsweisen Integration totaler Differentialgleichungen." *Zeitschrift für Mathematik und Physik* 46.

[126] Kästner, J. "Umbrella sampling." *WIREs Computational Molecular Science* 1, 6: (2011) 932–942. https://doi.org/10.1002/wcms.66.

[127] Kühne, T. D., M. Iannuzzi, M. Del Ben, V. V. Rybkin, P. Seewald, F. Stein, T. Laino, R. Z. Khaliullin, O. Schütt, F. Schiffmann, D. Golze, J. Wilhelm, S. Chulkov, M. H. Bani-Hashemian, V. Weber, U. Borštnik, M. Taillefumier, A. S. Jakobovits, A. Lazzaro, H. Pabst, T. Müller, R. Schade, M. Guidon, S. Andermatt, N. Holmberg, G. K. Schenter, A. Hehn, A. Bussy, F. Belleflamme, G. Tabacchi, A. Glöß, M. Lass, I. Bethune, C. J. Mundy, C. Plessl, M. Watkins, J. VandeVondele, M. Krack, and J. Hutter. "CP2K: An electronic structure and molecular dynamics software package - Quickstep: Efficient and accurate electronic structure calculations." *The Journal of Chemical Physics* 152, 19: (2020) 194,103. https://doi.org/10.1063/5.0007045.

[128] Laio, A., and M. Parrinello. "Escaping free-energy minima." *Proceedings of the National Academy of Sciences* 99, 20: (2002) 12,562–12,566. https://doi.org/10.1073/pnas.202427399.

[129] Larsen, A. H., J. J. Mortensen, J. Blomqvist, I. E. Castelli, R. Christensen, M. Dułak, J. Friis, M. N. Groves, B. Hammer, C. Hargus, E. D. Hermes, P. C. Jennings, P. B. Jensen, J. Kermode, J. R. Kitchin, E. L. Kolsbjerg, J. Kubal, K. Kaasbjerg, S. Lysgaard, J. B. Maronsson, T. Maxson, T. Olsen, L. Pastewka, A. Peterson, C. Rostgaard, J. Schiøtz, O. Schütt, M. Strange, K. S. Thygesen, T. Vegge, L. Vilhelmsen, M. Walter, Z. Zeng, and K. W. Jacobsen. "The atomic simulation environment—a Python library for working with atoms." *Journal of Physics: Condensed Matter* 29, 27: (2017) 273,002. https://doi.org/10.1088/1361-648X/aa680e.

[130] Leimkuhler, B., and C. Matthews. "Molecular Dynamics for MATLAB." http://www.moleculardynamics.info, .

[131] ———. "Rational Construction of Stochastic Numerical Methods for Molecular Sampling." *Applied Mathematics Research eXpress* 2013, 1: (2012) 34–56. https://doi.org/10.1093/amrx/abs010.

[132] ———. "Robust and efficient configurational molecular sampling via Langevin dynamics." *The Journal of Chemical Physics* 138, 17: (2013) 174,102. https://doi.org/10.1063/1.4802990.

[133] ———. "Efficient molecular dynamics using geodesic integration and solvent–solute splitting." *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 472, 2189. https://doi.org10.1098/rspa.2016.0138.

[134] Leimkuhler, B., and S. Reich. *Simulating Hamiltonian Dynamics*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005. https://doi.org/10.1017/CBO9780511614118.

[135] Leimkuhler, B. J., and C. Matthews. *Molecular Dynamics with Deterministic and Stochastic Numerical Methods*. Springer International Publishing, 2015. https://doi.org/10.1007/978-3-319-16375-8.

[136] Lemons, D. S., and A. Gythiel. "Paul Langevin's 1908 paper "On the Theory of Brownian Motion" ["Sur la théorie du mouvement brownien," C. R. Acad. Sci. (Paris) 146, 530–533 (1908)]." *American Journal of Physics* 65, 11: (1997) 1079–1081. https://doi.org/10.1119/1.18725.

[137] Li, Z., J. R. Kermode, and A. De Vita. "Molecular Dynamics with On-the-Fly Machine Learning of Quantum-Mechanical Forces." *Phys. Rev. Lett.* 114: (2015) 096,405. https://doi.org/10.1103/PhysRevLett.114.096405.

[138] Liang, T., Y. K. Shin, Y.-T. Cheng, D. E. Yilmaz, K. G. Vishnu, O. Verners, C. Zou, S. R. Phillpot, S. B. Sinnott, and A. C. van Duin. "Reactive Potentials for Advanced Atomistic Simulations." *Annual Review of Materials Research* 43, 1: (2013) 109–129. https://doi.org/10.1146/annurev-matsci-071312-121610.

[139] Lindemann, F. A. "The calculation of molecular vibration frequencies." *Phys. Z.* 11: (1910) 609.

[140] Lindorff-Larsen, K., S. Piana, R. O. Dror, and D. E. Shaw. "How Fast-Folding Proteins Fold." *Science* 334, 6055: (2011) 517–520. https://doi.org/10.1126/science.1208351.

[141] Lippert, G., J. Hutter, and M. Parrinello. "A hybrid Gaussian and plane wave density functional scheme." *Molecular Physics* 92, 3: (2010) 477–487. https://doi.org/10.1080/002689797170220.

[142] Louie, S. G., S. Froyen, and M. L. Cohen. "Nonlinear ionic pseudopotentials in spin-density-functional calculations." *Phys. Rev. B* 26: (1982) 1738–1742. http://link.aps.org/doi/10.1103/PhysRevB.26.1738.

[143] Mackerell, A. D., M. Feig, and C. L. Brooks. "Extending the treatment of backbone energetics in protein force fields: Limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations." *Journal of Computational Chemistry* 25, 11: (2004) 1400–1415. https://doi.org/10.1002/jcc.20065.

[144] Maragliano, L., and E. Vanden-Eijnden. "A temperature accelerated method for sampling free energy and determining reaction pathways in rare events simulations." *Chemical Physics Letters* 426, 1: (2006) 168 – 175. https://doi.org/10.1016/j.cplett.2006.05.062.

[145] Marinari, E., and G. Parisi. "Simulated Tempering: A New Monte Carlo Scheme." *Europhysics Letters (EPL)* 19, 6: (1992) 451–458. https://doi.org/10.1209%2F0295-5075%2F19%2F6%2F002.

[146] Martinsson, A., J. Lu, B. Leimkuhler, and E. Vanden-Eijnden. "The simulated tempering method in the infinite switch limit with adaptive weight learning." *Journal of Statistical Mechanics: Theory and Experiment* 2019, 1: (2019) 013,207. https://doi.org/10.1088%2F1742-5468%2Faaf323.

[147] Martyna, G. J., M. L. Klein, and M. Tuckerman. "Nosé–Hoover chains: The canonical ensemble via continuous dynamics." *The Journal of Chemical Physics* 97, 4: (1992) 2635–2643. https://doi.org/10.1063/1.463940.

[148] Martyna, G. J., M. E. Tuckerman, D. J. Tobias, and M. L. Klein. "Explicit reversible integrators for extended systems dynamics." *Molecular Physics* 87, 5: (1996) 1117–1157. https://doi.org/10.1080/00268979600100761.

[149] Matsui, T., and M. H. Manghnani. "Thermal expansion of single-crystal forsterite to 1023 K by Fizeau interferometry." *Physics and Chemistry of Minerals* 12, 4: (1985) 201–210. https://doi.org/10.1007/BF00311289.

[150] Message Passing Interface Forum. "MPI: A Message-Passing Interface Standard Version 3.1." Technical report, 2015. https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[151] Michalis, V. K., I. N. Tsimpanogiannis, A. K. Stubos, and I. G. Economou. "Direct phase coexistence molecular dynamics study of the phase equilibria of the ternary methane–carbon dioxide–water hydrate system." *Phys. Chem. Chem. Phys.* 18: (2016) 23,538–23,548. https://doi.org/10.1039/C6CP04647A.

[152] Misic, M., I. Bethune, and M. Tomasevic. "Automated regression testing and code coverage analysis of the CP2K application." In *Proceedings of the 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. 2014. https://doi.org/10.1109/ICST.2014.21.

[153] Miyamoto, S., and P. A. Kollman. "Settle: An analytical version of the SHAKE and RATTLE algorithm for rigid water models." *Journal of Computational Chemistry* 13, 8: (1992) 952–962. https://doi.org/10.1002/jcc.540130805.

[154] Morrone, J. A., T. E. Markland, M. Ceriotti, and B. J. Berne. "Efficient multiple time scale molecular dynamics: Using colored noise thermostats to stabilize resonances." *The Journal of Chemical Physics* 134, 1: (2011) 014,103. https://doi.org/10.1063/1.3518369.

[155] Morse, P. M. "Diatomic Molecules According to the Wave Mechanics. II. Vibrational Levels." *Phys. Rev.* 34: (1929) 57–64. https://doi.org/10.1103/PhysRev.34.57.

[156] Mortensen, J. J., L. B. Hansen, and K. W. Jacobsen. "Real-space grid implementation of the projector augmented wave method." *Phys. Rev. B* 71: (2005) 035,109. https://doi.org/10.1103/PhysRevB.71.035109.

[157] Mueller, T., A. Hernandez, and C. Wang. "Machine learning for interatomic potential models." *The Journal of Chemical Physics* 152, 5: (2020) 050,902. https://doi.org/10.1063/1.5126336.

[158] Mulliken, R. S. "Electronic Population Analysis on LCAO–MO Molecular Wave Functions. I." *The Journal of Chemical Physics* 23, 10: (1955) 1833–1840. https://doi.org/10.1063/1.1740588.

[159] Nguyen, P. H., Y. Okamoto, and P. Derreumaux. "Communication: Simulated tempering with fast on-the-fly weight determination." *The Journal of Chemical Physics* 138, 6: (2013) 061,102. https://doi.org/10.1063/1.4792046.

[160] Nosé, S. "A unified formulation of the constant temperature molecular dynamics methods." *The Journal of Chemical Physics* 81, 1: (1984) 511. https://doi.org/10.1063/1.447334.

[161] ———. "A molecular dynamics method for simulations in the canonical ensemble." *Molecular Physics* 52, 2: (1984) 255–268. https://doi.org/10.1080/00268978400101201.

[162] Omeltchenko, A., T. Campbell, R. Kalia, X. Liu, A. Nakano, and P. Vashishta. "Scalable I/O of large-scale molecular dynamics simulations: A data-compression algorithm." *Computer Physics Communications* 131: (2000) 78–85. https://doi.org/10.1016/S0010-4655(00)00083-7.

[163] OpenMP Architecture Review Board. "OpenMP 5.0 Specification." Technical report, 2018. https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf.

[164] Parrinello, M., and A. Rahman. "Crystal Structure and Pair Potentials: A Molecular-Dynamics Study." *Phys. Rev. Lett.* 45: (1980) 1196–1199. https://doi.org/10.1103/PhysRevLett.45.1196.

[165] ———. "Polymorphic transitions in single crystals: A new molecular dynamics method." *Journal of Applied Physics* 52, 12: (1981) 7182–7190. https://doi.org/10.1063/1.328693.

[166] Pashov, D., S. Acharya, W. R. Lambrecht, J. Jackson, K. D. Belashchenko, A. Chantis, F. Jamet, and M. van Schilfgaarde. "Questaal: A package of electronic structure methods based on the linear muffin-tin orbital technique." *Computer Physics Communications* 249: (2020) 107,065. https://doi.org/10.1016/j.cpc.2019.107065.

[167] Payne, M., M. Teter, D. Allan, T. Arias, and J. Joannopoulos. "Iterative minimization techniques for ab initio total-energy calculations: molecular dynamics and conjugate gradients." *Reviews of Modern Physics* 64, 4: (1992) 1045–1097. https://doi.org/10.1103/RevModPhys.64.1045.

[168] Pedone, A., G. Malavasi, M. C. Menziani, A. N. Cormack, and U. Segre. "A new self-consistent empirical interatomic potential model for oxides, silicates, and silica-based glasses." *The Journal of Physical Chemistry. B* 110, 24: (2006) 11,780–95. https://doi.org/10.1021/jp0611018.

[169] Perdew, J. P., K. Burke, and M. Ernzerhof. "Generalized Gradient Approximation Made Simple." *Physical Review Letters* 77, 18: (1996) 3865–3868. https://doi.org/10.1103/PhysRevLett.77.3865.

[170] Phillips, J. C., Gengbin Zheng, S. Kumar, and L. V. Kale. "NAMD: Biomolecular Simulation on Thousands of Processors." In *SC '02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. 2002, 36–36. 10.1109/SC.2002.10019.

[171] Phillips, J. C., R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. "Scalable molecular dynamics with NAMD." *Journal of Computational Chemistry* 26, 16: (2005) 1781–1802. https://doi.org/10.1002/jcc.20289.

[172] Piana, S., K. Lindorff-Larsen, R. M. Dirks, J. K. Salmon, R. O. Dror, and D. E. Shaw. "Evaluating the Effects of Cutoffs and Treatment of Long-range Electrostatics in Protein Folding Simulations." *PLOS ONE* 7, 6: (2012) 1–6. https://doi.org/10.1371/journal.pone.0039918.

[173] Plattner, N., S. Doerr, G. De Fabritiis, and F. Noé. "Complete protein-protein association kinetics in atomic detail revealed by molecular dynamics simulations and Markov modelling." *Nature chemistry* 9, 10: (2017) 1005—1011. https://doi.org/10.1038/nchem.2785.

[174] Plimpton, S. "Fast Parallel Algorithms for Short-Range Molecular Dynamics." *Journal of Computational Physics* 117, 1: (1995) 1 – 19. https://doi.org/10.1006/jcph.1995.1039.

[175] Posch, H. A., W. G. Hoover, and F. J. Vesely. "Canonical dynamics of the Nosé oscillator: Stability, order, and chaos." *Phys. Rev. A* 33: (1986) 4253–4265. https://doi.org/10.1103/PhysRevA.33.4253.

[176] Preto, J., and C. Clementi. "Fast recovery of free energy landscapes via diffusion-map-directed molecular dynamics." *Phys. Chem. Chem. Phys.* 16: (2014) 19,181–19,191. https://doi.org/10.1039/C3CP54520B.

[177] Pulay, P. "Ab initio calculation of force constants and equilibrium geometries in polyatomic molecules." *Molecular Physics* 17, 2: (1969) 197–204. https://doi.org/10.1080/00268976900100941.

[178] Richardson, A. "GPU Acceleration of HPC Applications." http://static.epcc.ed.ac.uk/dissertations/hpc-msc/2008-2009/Alan_Richardson.pdf, 2009.

[179] Richet, P., P. Gillet, and A. Pierre. "Raman spectroscopy, x-ray diffraction, and phase relationship determinations with a versatile heating cell for measurements up to 3600 K (or 2700 K in air)." *Journal of Applied Physics* 74, 9: (1993) 5451–5456. https://doi.org/10.1063/1.354256.

[180] Roeder, P., and R. Emslie. "Olivine-liquid equilibrium." *Contributions to mineralogy and petrology* 29: (1970) 275–289. http://link.springer.com/article/10.1007/BF00371276.

[181] Rowe, P., V. L. Deringer, P. Gasparotto, G. Csányi, and A. Michaelides. "An accurate and transferable machine learning potential for carbon." *The Journal of Chemical Physics* 153, 3: (2020) 034,702. https://doi.org/10.1063/5.0005084.

[182] Ryckaert, J.-P., G. Ciccotti, and H. J. Berendsen. "Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes." *Journal of Computational Physics* 23, 3: (1977) 327 – 341. https:/doi.org/10.1016/0021-9991(77)90098-5.

[183] Salomon Ferrer, R., A. Götz, D. Poole, S. Grand, and R. Walker. "Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald." *Journal of Chemical Theory and Computation* 9: (2013) 3878–3888. https://doi.org/10.1021/ct400314y.

[184] Sanloup, C., J. Drewitt, C. Crépisson, Y. Kono, C. Park, C. McCammon, L. Hennet, S. Brassamin, and a. Bytchkov. "Structure and density of molten fayalite at high pressure." *Geochimica et Cosmochimica Acta* 118: (2013) 118–128. https://doi.org/10.1016/j.gca.2013.05.012.

[185] Sanloup, C., J. W. E. Drewitt, Z. Konôpková, P. Dalladay-Simpson, D. M. Morton, N. Rai, W. van Westrenen, and W. Morgenroth. "Structural change in molten basalt at deep mantle conditions." *Nature* 503, 7474: (2013) 104–7. https://doi.org/10.1038/nature12668.

[186] Senftle, T., S. Hong, M. M. Islam, S. Kylasa, Y. Zheng, Y. K. Shin, C. Junkermeier, R. Engel-Herbert, M. J. Janik, H. M. Aktulga, T. Verstraelen, A. Grama, and A. C. T. van Duin. "The ReaxFF reactive force-field: Development, applications, and future directions." *npj Computational Materials* 2. https://doi.org/10.1038/npjcompumats.2015.11.

[187] Shanno, D. "Conditioning of quasi-Newton methdos for function minimization." *Mathematics of Computation* 24.

[188] Shaw, D. E., M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ierardi, I. Kolossváry, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang. "Anton, a Special-purpose Machine for Molecular Dynamics Simulation." *Commun. ACM* 51, 7: (2008) 91–97. https://doi.org/10.1145/1364782.1364802.

[189] Shaw, D. E., P. Maragakis, K. Lindorff-Larsen, S. Piana, R. O. Dror, M. P. Eastwood, J. A. Bank, J. M. Jumper, J. K. Salmon, Y. Shan, and W. Wriggers. "Atomic-Level Characterization of the Structural Dynamics of Proteins." *Science* 330, 6002: (2010) 341–346. https://doi.org/10.1126/science.1187409.

[190] Shinoda, W., M. Shiga, and M. Mikami. "Rapid estimation of elastic constants by molecular dynamics simulation under constant stress." *Phys. Rev. B* 69: (2004) 134,103. https://doi.org/10.1103/PhysRevB.69.134103.

[191] Shkurti, A., I. Styliari, V. Balasubramanian, I. Bethune, C. Pedebos, S. Jha, and C. Laughton. "CoCo-MD: A Simple and Effective Method for the Enhanced Sampling of Conformational Space." *Journal of Chemical Theory and Computation* 15, 4: (2019) 2587–2596. https://doi.org/10.1021/acs.jctc.8b00657.

[192] Shvets, P., V. Voevodin, and S. Zhumatiy. "Statistics of Software Package Usage in Supercomputer Complexes." 2017.

[193] Singh, D. *Planewaves, pseudopotentials and the LAPW method*. Kluwer Academic, 1994. https://10.1007/978-0-387-29684-5.

[194] Smyth, J. R. "A simple heating stage for single-crystal diffraction studies up to 1000 C." *American Mineralogist* 57: (1972) 1305–1309. http://www.minsocam.org/ammin/AM47/AM47_1431.pdf.

[195] Snow, C. D., H. Nguyen, V. S. Pande, and M. Gruebele. "Absolute comparison of simulated and experimental protein-folding dynamics." *Nature* 420, 6911: (2002) 102—106. https://doi.org/10.1038/nature01160.

[196] Soares, T. A., P. H. Hünenberger, M. A. Kastenholz, V. Kräutler, T. Lenz, R. D. Lins, C. Oostenbrink, and W. F. van Gunsteren. "An improved nucleic acid parameter set for the GROMOS force field." *Journal of Computational Chemistry* 26, 7: (2005) 725–737. https://doi.org/10.1002/jcc.20193.

[197] Stixrude, L., E. Wasserman, and R. E. Cohen. "Composition and temperature of Earth's inner core." *Journal of Geophysical Research: Solid Earth* 102, B11: (1997) 24,729–24,739. https://doi.org/10.1029/97JB02125.

[198] Stone, A. J. "ORIENT version 4.9." http://www-stone.ch.cam.ac.uk/documentation/orient/Manual.pdf, 2018.

[199] Sugita, Y., and Y. Okamoto. "Replica-exchange molecular dynamics method for protein folding." *Chemical Physics Letters* 314, 1: (1999) 141 – 151. https://doi.org/10.1016/S0009-2614(99)01123-9.

[200] Suzuki, I., O. Anderson, and Y. Sumino. "Elastic properties of a single-crystal forsterite Mg2SiO4, up to 1,200 K." *Physics and Chemistry of Minerals* 38–46. https://doi.org/10.1007/BF01204324.

[201] Tangney, P., and S. Scandolo. "An ab initio parametrized interatomic force field for silica." *The Journal of Chemical Physics* 117: (2002) 8898. https://doi.org/10.1063/1.1513312.

[202] The CP2K Developers Group. "CP2K: Open Source Molecular Dynamics.", . http://www.cp2k.org.

[203] Todorov, I. T., W. Smith, K. Trachenko, and M. T. Dove. "DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism." *J. Mater. Chem.* 16: (2006) 1911–1918. https://doi.org/10.1039/B517931A.

[204] Todorov, I., I. Bush, and A. Porter. "The need for parallel I/O in classical molecular dynamics." 2008. https://cug.org/5-publications/proceedings_attendee_lists/2008CD/S08_Proceedings/pages/Authors/01-5Monday/Todorov-Monday3B/Todorov-Monday3B-paper.pdf.

[205] Tribello, G. A., M. Bonomi, D. Branduardi, C. Camilloni, and G. Bussi. "PLUMED 2: New feathers for an old bird." *Computer Physics Communications* 185, 2: (2014) 604 – 613. https://doi.org/10.1016/j.cpc.2013.09.018.

[206] Tuckerman, M., B. J. Berne, and G. J. Martyna. "Reversible multiple time scale molecular dynamics." *The Journal of Chemical Physics* 97, 3: (1992) 1990–2001. https://doi.org/10.1063/1.463137.

[207] Tuckerman, M., J. Alejandre, R. López-Rendón, A. Jochim, and G. Martyna. "A Liouville-operator derived measure-preserving integrator for molecular dynamics simulations in the isothermal-isobaric ensemble." *Journal of Physics A: Mathematical and Theoretical* 39, 19: (2006) 5629–5651. https://doi.org/10.1088/0305-4470/39/19/S18.

[208] Magdău, I. B., and G. J. Ackland. "Identification of high-pressure phases III and IV in hydrogen: Simulating Raman spectra using molecular dynamics." *Phys. Rev. B* 87: (2013) 174,110. https://doi.org/10.1103/PhysRevB.87.174110.

[209] Vanderbilt, D. "Soft self-consistent pseudopotentials in a generalized eigenvalue formalism." *Phys. Rev. B* 41: (1990) 7892–7895. https://doi.org/10.1103/PhysRevB.41.7892.

[210] VandeVondele, J., and J. Hutter. "An efficient orbital transformation method for electronic structure calculations." *The Journal of Chemical Physics* 118, 10: (2003) 4365–4369. https://doi.org/10.1063/1.1543154.

[211] VandeVondele, J., M. Krack, F. Mohamed, M. Parrinello, T. Chassaing, and J. Hutter. "Quickstep: Fast and accurate density functional calculations using a mixed Gaussian and plane waves approach." *Computer Physics Communications* 167, 2: (2005) 103 – 128. https://doi.org/10.1016/j.cpc.2004.12.014.

[212] Vega, C., M. M. Conde, C. McBride, J. L. F. Abascal, E. G. Noya, R. Ramirez, and L. M. Sesé. "Heat capacity of water: A signature of nuclear quantum effects." *The Journal of Chemical Physics* 132, 4: (2010) 046,101. https://doi.org/10.1063/1.3298879.

[213] Verlet, L. "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules." *Phys. Rev.* 159: (1967) 98–103. https://doi.org/10.1103/PhysRev.159.98.

[214] Voelz, V. A., G. R. Bowman, K. Beauchamp, and V. S. Pande. "Molecular simulation of ab initio protein folding for a millisecond folder NTL9(1-39)." *Journal of the American Chemical Society* 132, 5: (2010) 1526–1528. https://doi.org/10.1021/ja9090353. PMID: 20070076.

[215] Walker, A. M., K. Wright, and B. Slater. "A computational study of oxygen diffusion in olivine." *Physics and Chemistry of Minerals* 30, 9: (2003) 536–545. https://doi.org/10.1007/s00269-003-0358-7.

[216] Wegener, A. "Die Herausbildung der Grossformen der Erdrinde (Kontinente und Ozeane), auf geophysikalischer Grundlage." *Petermanns Geographische Mitteilungen* 63.

[217] Wilson, J. T. "A New Class of Faults and their Bearing on Continental Drift." *Nature* 207. https://doi.org/10.1038/207343a0.

[218] Wu, H., A. S. J. S. Mey, E. Rosta, and F. Noé. "Statistically optimal analysis of state-discretized trajectory data from multiple thermodynamic states." *The Journal of Chemical Physics* 141, 21: (2014) 214,106. https://doi.org/10.1063/1.4902240.

[219] Yoshida, H. "Construction of higher order symplectic integrators." *Physics Letters A* 150, 5: (1990) 262 – 268. https://doi.org/10.1016/0375-9601(90)90092-3.

[220] Zhang, T., P. H. Nguyen, and P. Derreumaux. "Simulated Tempering for GROMACS." http://www-lbt.ibpc.fr/sites/default/files/u76/ALA10_ST.zip, .

[221] Zhang, T., P. H. Nguyen, J. Nasica-Labouze, Y. Mu, and P. Derreumaux. "Folding Atomistic Proteins in Explicit Solvent Using Simulated Tempering." *The Journal of Physical Chemistry B* 119, 23: (2015) 6941–6951. https://doi.org/10.1021/acs.jpcb.5b03381.

[222] Zheng, W., M. A. Rohrdanz, and C. Clementi. "Rapid Exploration of Configuration Space with Diffusion-Map-Directed Molecular Dynamics." *The Journal of Physical Chemistry B* 117, 42: (2013) 12,769–12,776. https://doi.org/10.1021/jp401911h. PMID: 23865517.