Algorithms for adaptively restrained molecular dynamics

Krishna Kant Singh

Communauté UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES

Spécialité : Informatique

Arrêté ministériel : 25 mai 2016

Présentée par

Krishna kant SINGH

Thèse dirigée par **Jean-Francois MEHAUT**, Professeur, Université Grenoble Alpes, et
codirigée par **Stephane REDON** CR INRIA
préparée au sein du **Laboratoire Laboratoire Jean Kuntzmann** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

Algorithmes pour la dynamique moléculaire restreinte de manière adaptative

Algorithms for Adaptively Restrained Molecular Dynamics

Thèse soutenue publiquement le **8 novembre 2017**,
devant le jury composé de :

Monsieur JEAN-FRANÇOIS MEHAUT
PROFESSEUR, UNIVERSITE GRENOBLE ALPES, Directeur de thèse
Monsieur STEPHANE REDON
DIRECTEUR DE RECHERCHE, INRIA DELEGATION ALPES, Co-directeur de thèse
Monsieur OLIVIER COULAUD
DIRECTEUR DE RECHERCHE, INRIA CENTRE BORDEAUX-SUD-OUEST, Rapporteur
Monsieur KONRAD HINSEN
CHERCHEUR, CNRS CENTRE-LIMOUSIN POITOU-CHARENTES, Rapporteur
Monsieur JEAN-LOUIS BARRAT
PROFESSEUR, UNIVERSITE GRENOBLE ALPES, Président
Monsieur MARC BAADEN
CHARGE DE RECHERCHE, CNRS DELEGATION PARIS, Examinateur
Monsieur BENJAMIN BOUVIER
CHARGE DE RECHERCHE, CNRS NORD-PAS DE CALAIS ET PICARDIE, Examinateur

Publications

Krishna Kant Singh and Stephane Redon, *Adaptively Restrained Molecular Dynamics in LAMMPS*, Modelling and Simulation in Materials Science and Engineering, 2017, DOI:10.1088/1361- 651X/aa7345.
Krishna Kant Singh, Dmitriy F. Marin and S. Redon, *Parallel Adaptively Restrained Molecular Dynamics*, International Conference on High Performance Computing and Simulation (HPCS 2017).
Krishna Kant Singh and Stephane Redon, *Single-pass Incremental Force Updates for Adaptively Restrained Molecular Dynamics*,Journal of Computational Chemistry, 2017, DOI: 10.1002/jcc.25126

Talk

International Conference on High Performance Computing and Simulation (HPCS 2017), Genova

Posters

Euromech-conference 2016, INRIA Montbonnot, France.
ARC-conference 2015, Grenoble, France.

Acknowledgements

It is a pleasure to acknowledge and thank people who have been helpful throughout my journey towards PhD. I would like to express my gratitude to my advisor, Dr. Stephane Redon. His knowledge, enthusiasm and guidance have helped me in my research as well as in writing of this thesis. I consider him not only as a supervisor, but also as great human being. I would also like to acknowledge my another supervisor Dr. Jean-Francois MEHAUT for providing clusters and resources for performing simulations.

I would also like to my thesis reviewers for their valuable suggestions and the members of the jury for their questions and comments.

I would like to thank NANO-D team-mates Sergei Grudinin, Zofia Trstanova, Svetlana Artemova, Léonard Jaillet, Petr Popov, Semeho Edorh, Emilie Neuveu, Khoa Nguyen, Guillaume Pages, François Rousse, Alexandre Hoffmann, Jocelyn Gaté, Maria and others for their constant support, encouragement, cakes and helping me to enhance my knowledge in various fields including french.

I would also like to thank Imma Presseguer and Julie Bourget for their assistance in all the administrative and non-administrative works.

I also take this opportunity to thank INRIA for providing an opportunity to see the world of research so closely.

Beside that, I want to thank my friends Dharam, Fabian, Naweiluo, Mandy, Bipin, Broto and Farheen for their constant support.

This acknowledgement would not be complete without mentioning the support of my parents, I am very much grateful to my parents for their love and support throughout my studies.

Abstract

Molecular Dynamics (MD) is often used to simulate large and complex systems. Although, simulating such complex systems for the experimental time scales are still computationally challenging. In fact, the most computationally extensive step in MD is the computation of forces between particles. Adaptively Restrained Molecular Dynamics (ARMD) is a recently introduced particles simulation method that switches positional degrees of freedom on and off during simulation. Since force computations mainly depend upon the inter-atomic distances, the force computation between particles with positional degrees of freedom off (restrained particles) can be avoided. Forces involving active particles (particles with positional degrees of freedom on) are computed. In order to take advantage of adaptability of ARMD, we designed novel algorithms to compute and update forces efficiently. We designed algorithms not only to construct neighbor lists, but also to update them incrementally. Additionally, we designed single-pass incremental force update algorithm that is almost two times faster than previously designed two-pass incremental algorithm. These proposed algorithms are implemented and validated in the LAMMPS MD simulator, however, these algorithms can be applied to other MD simulators. We assessed our algorithms on different and diverse benchmarks in both microcanonical ensemble (NVE) and canonical (NVT) ensembles. In the NVE ensemble, ARMD allows users to trade between precision and speed while, in the NVT ensemble, it makes it possible to compute statistical averages faster. In Last, we introduce parallel algorithms for single-pass incremental force computations to take advantage of adaptive restraints using the Message Passage Interface (MPI) standard.

Abstract

Les méthodes de dynamique moléculaire (MD pour Molecular Dynamics en anglais) sont utilisées pour simuler des systèmes volumineux et complexes. Cependant, la simulation de ce type de systèmes sur de longues échelles temporelles demeure un problème coûteux en temps de calcul. L'étape la plus coûteuse des méthodes de MD étant la mise à jour des forces entre les particules. La simulation de particules restreintes de façon adaptative (ARMD pour Adaptively Restrained Molecular Dynamics en anglais) est une nouvelle approche permettant d'accélérer le processus de simulation en réduisant le nombre de calculs de forces effectués à chaque pas de temps. La méthode ARMD fait varier l'état des degrés de liberté en position en les activants ou en les désactivants de façon adaptative au cours de la simulation. Du fait, que le calcul des forces dépend majoritairement de la distance entre les atomes, ce calcul peut être évité entre deux particules dont les degrés de liberté en position sont désactivés. En revanche, le calcul des forces pour les particules actives (i.e. celles dont les degrés de liberté en position sont actifs) est effectué. Afin d'exploiter au mieux l'adaptabilité de la méthode ARMD, nous avons conçu de nouveaux algorithmes permettant de calculer et de mettre à jour les forces de façon plus efficace. Nous avons développé des algorithmes permettant de construire et de mettre à jour des listes de voisinage de manière incrémentale. En particulier, nous avons travaillé sur un algorithme de mise à jour incrémentale des forces en un seul passage deux fois plus rapide que l'ancien algorithme également incrémental mais qui nécessitait deux passages. Les méthodes proposées ont été implémentées et validées dans le simulateur de MD appelé LAMMPS, mais elles peuvent s'appliquer à n'importe quel autre simulateur de MD. Nous avons validé nos algorithmes pour différents exemples sur les ensembles NVE et NVT. Dans l'ensemble NVE, la méthode ARMD permet à l'utilisateur de jouer sur le précision pour accélérer la vitesse de la simulation. Dans l'ensemble NVT, elle permet de mesurer des grandeurs statistiques plus rapidement. Finalement, nous présentons des algorithmes parallèles pour la mise à jour incrémentale en un seul passage permettant d'utiliser la méthode ARMD avec le standard Message Passage Interface (MPI).

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Scientific studies are conducted with the aim of obtaining a deeper insight into the naturally occurring phenomena, using knowledge from different fields like physics, chemistry, mathematics, biology among others. To understand the different phenomena, experiments are conducted in a controlled environment (*in vivo* or *in vitro*) at laboratories, where detailed studies are carried out. The ultimate goal of these experiments is to deduct a mathematical description of the observed phenomenon in order to predict and analyze different properties of a system under various environmental conditions.

In some cases, the experimental conditions may not be conducive for studying certain physical properties in detail (for instance, study of material at very high pressure, or effect of a cavity in material), either due to the complexity of a system or limitations posed by the experimental technique itself. Computer simulations can aid in overcoming the aforementioned limitations by solving a representative mathematical model (a set of equations). For example, Navier-Stokes equations are frequently used in fluid mechanics; however, analytical solutions of these equations are difficult to obtain. The Navier-Stokes equation is solved by discretization and the discretized solution of Navier-Stokes equations is then used to simulate fluid systems using computers.

Systems in other domains such as astronomy, biology, and material science, often use computers to solve representative equations and to study the dynamical behaviour of a system [4–7]. In these domains, a system can be represented with a set of particles or points and forces on these particles are exerted by other particles based on certain rules. This representation of a system is referred as the particle model. For example, in astronomy, planets are represented as points, and interactions between these planets are governed by gravitational laws. These points are also called interacting bodies, and dynamics of a system can be solved analytically for few (2-3) interacting bodies; however, in nature, there are millions of interacting bodies, and obtaining a solution for such large systems is very

complicated. The presence of astronomical number of particles and complex experimental set up are among the main reasons that have accounted for the rise of computer simulations in science.

In biology, computer simulations may serve as a computational microscope, due to the ability to investigate the biomolecules with atomistic details [8, 9]. By providing the positions of particles as a function of time, in particular, computer simulations help to rationalize the behaviour of biological molecules, to gain insight into the folding mechanism of small proteins, structural-functional relationship of biomolecules, the selectivity of ion-channels etc. [10–22]. Apart from the structural properties of biomolecules, computer simulations are used to decipher the dynamical properties of biomolecules and to gain atomistic level details of the malfunctioning of proteins that might lead to a disease (in Alzheimer's disease, specific proteins change their conformations from alpha-helices to beta-sheets) [3, 23–26]. Moreover, computer simulations are often used to characterized the rate of ligand association and dissociation mechanism [27–29]. These simulations are used to assist experimentalists in designing new drugs, chemical-compounds and biosensors etc.. In nanotechnology, scientists often conduct experiments and simulations to study the influence of a biomolecule (DNA or protein) on the properties of graphene, and these informations are useful to design new biosensors that can detect a disease [30–33]. Computer simulations are also used to compute properties of solids and liquids such as melting, boiling temperature point and effect of impurities on the melting temperature. Thus, Computer simulations act as a bridge between theoretician and experimentalist.

1.0.1 Representative Particle Models

Most often, in the simulation, a system is represented by a computer model and the dynamics of this model is evolved by solving an appropriate equation. These representative models can vary in the levels of detail (fig. 1.1). Based on the level of detail, computer simulations can be categorised as:

1. **Quantum mechanics (QM)**

   In the QM simulation, a system is represented at the quantum mechanics level. The QM simulation is used to study processes that require information about electrons [34]. The QM simulations are performed by solving the Schrödinger wave equation. Moreover, the Born–Oppenheimer (BO) approximation can be used to decouple the electronic and the nuclear degrees of freedom.

Solving Schrödinger wave equations (time-dependent and time-independent) are computationally expensive, therefore, the QM method can be used to simulate a very small system that involves up to hundreds of atoms. The number of atoms, however, can be increased by using various level of approximations such as Hartree–Fock, density functional theory (DFT) or Car–Parrinello molecular dynamics [35–37]. QM simulations are generally used, but not limited to, in simulating bond breaking and formation, reactions involving charge transfer and structures of small molecules etc. QM has been used to simulate plethora of small molecules e.g. $H_2^+, HD^+, H3^+$, and to compute properties like proton–proton coupling constants of organic molecules such as methylene [38, 39]. These types of simulations are useful in obtaining precise information of a system such as structure of methylene, water (bond length and angle), proton transfers in photo-receptor proteins [40]. However, owing to the huge computational need, this method is limited to small systems.

2. Classical Mechanics simulation

This simulation method is used to study processes that do not require any quantum level details (e.g. transfer of proton or electron or bond breaking and formation). Methods based on this mechanics ignore the electronic degrees of freedom and particles are represented as the positions of atomic nuclei. Moreover, a system containing these particles follows the classical mechanics theory (Newton's law of motions). The system can be simulated using either MC or MD. One of the main objective of these methods is to sample configurations over the phase space so that correct averages and thermodynamic properties can be obtained. MC is a probabilistic method, in this configurations are randomly generated, and then based on certain criteria (depend upon a given temperature) a generated configuration is either accepted or rejected. These criteria ensure to generate low energy configurations with high probability as compared to configurations correspond to higher energy values. On the other hand, MD is a deterministic simulation method that generate configurations as a function of time. These simulation methods along with physical parameters (temperature, force fields and either equation of motions or metropolis) generate configurations such that physiochemical properties of a system can be obtained. In particular, MD is used to decipher the structural-functional relationship of biomolecules and provide atomistic level details about conformational changes associated with biomolecules. Moreover, MD is also used in conjugation with experimental techniques (such as X-ray and NMR) to determine and to refine structures of biomolecules.

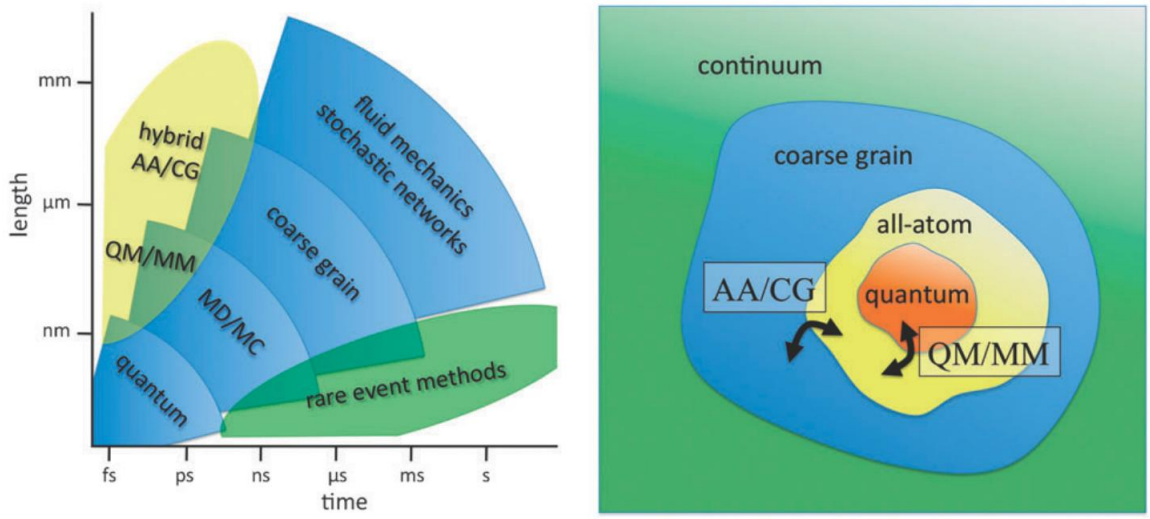


Fig. 1.1 This figure shows simulation methods at different length and time scales. High accuracy quantum simulations can be used to simulate a small system up to *ps* time scale. Whereas, simulations with low level of accuracy (coarse grain) can be used to simulate systems with μm and up to *ms* time scale. Picture taken from Ref [1].

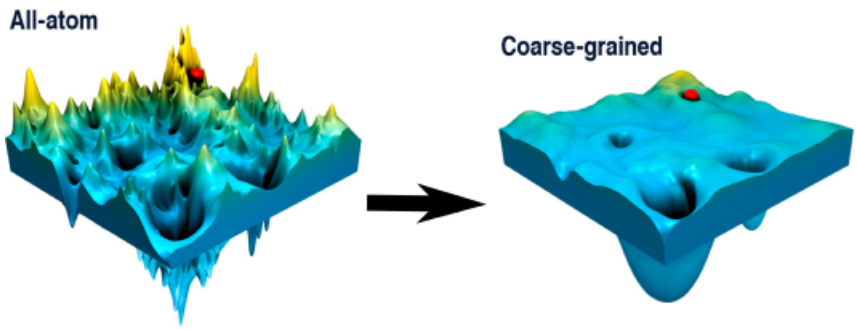


Fig. 1.2 This figure illustrate the energy surface of all-atom and coarse-grained simulations along order parameters. All-atom simulation has a very rugged energy surface; hence, simulation can be performed by using a short time step ($1-2\ fs$). On the contrary, the energy surface associated with a coarse-grained simulation is smooth, thus a larger time step size can be used to simulate a system. Picture taken from Ref [2].

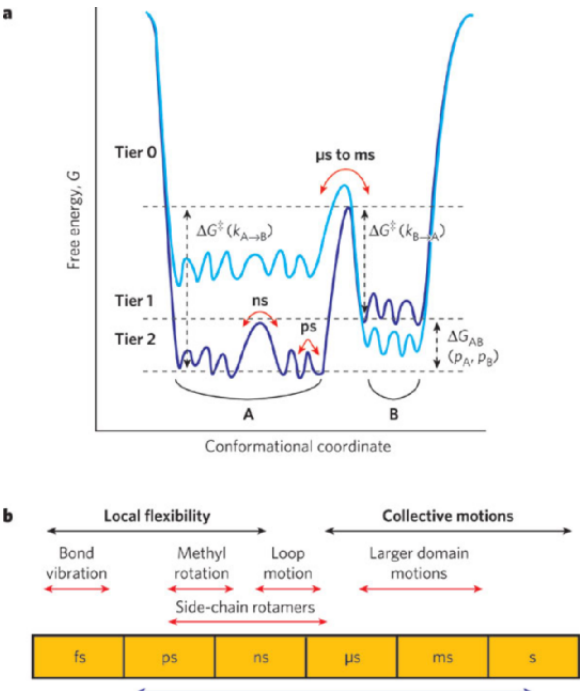


Fig. 1.3 a) This Figure represents the one-dimensional energy landscape of a protein and the energy barriers. Simulations at a time scale of $ps - ns$ are useful to study local flexibilities of biomolecule (bond vibration, methyl rotation and loop motions). Owing to the rugged energy landscape of a biomolecule, most often, simulations explore regions which are separated by lower free energy barriers (up to few k_bT). In order to sample regions that are separated by high energy barriers, one need to perform longer simulations ($\mu s - ms$). b) Time scale of protein dynamics. Most biological relevant events like collective motions, protein folding, ligand association and dissociation occur at μs to s time scale. Picture taken from Ref [3].

Molecular simulations have been used to simulate small and simple systems such as water and butane to complex and large biological systems e.g. cell-membrane, ion-channel and ribosome. In simulations of biological molecules, because of the fast H-bond vibrations, all-atom simulations are perform by using a time step of 1 fs or 2 fs by using a constrained algorithm (SHAKE or RATTLE) [41, 42]. The time-step can be further increased up to $4-5fs$ by either using virtual interaction sites, which removes fast H-bond vibrations or highest frequency vibrations present in the system, or Hydrogen-Mass-Repartition [43, 44]. Due to the short time step size, all-atom simulations are currently limited to microsecond time scales (fig. 2). In fact, for complex systems such as cell membrane proteins or crowding, owing to the number of force computations at each step and short time step, molecular dynamics may suffer in the performance and may not be able to sample high-energy conformations associated with a biomolecule. In order to increase the time step size and study biomolecules beyond the microsecond time-scale, coarse-grained simulations can be performed.

3. Coarse-grained (CG) simulation

The CG representation involves reducing the levels of detail of atoms by grouping them into coarse-grained groups (also called CG-beads), and hence reducing the number of atoms present in the system. For example, one water molecule can be represented by a

one CG-bead. Moreover, CG simulations also smooth the free energy surface from a rugged free-energy landscapes, as a result, a larger time-step ($10-100\ fs$) can be used in performing a CG simulation of a system (fig. 1.2). CG simulations are often used in simulation of complex systems involving proteins, biological lipid-membrane and crowding. Although a CG representation significantly increases the speed of a simulation, it reduces the level of accuracy.

4. **Supra-coarse-grained simulation**

   This representation reduces the levels of detail further by representing multiple CG beads into one supra bead, for example, one supra-coarse-grained bead of water might contain 4-5 water molecules. This representation significantly reduces the number of atoms present in a system, and thus simulations can be perform faster [45].

1.0.2 Hybrid representations

Apart from these categories, a hybrid representation of a system is also use in computer simulations. This representation incorporates different levels of detail. Precisely, a specific part of the system is simulated at a different scale, whereas other scale has been used for the rest of the system. These hybrid simulations are:

1. **QM/MM simulations**

   This simulation combines quantum mechanics simulations with molecular mechanics. In QM/MM method, a small and interesting part of a system is treated at an appropriate level of quantum chemistry theory, while the rest of the system is treated with the fast molecular mechanics method. The QM/MM simulation method may be used to study systems such as interactions of a drug with proteins, protein-ligand complexes and photoreceptor proteins [46, 47].

2. **MM/CG simulations**

   In this hybrid approach, a specific part of the system is represented with all-atoms and the rest of the system is treated with CG beads. This representation is frequently use in simulation of a proteins in a lipid membrane, where the protein is represented with all-atom and lipid molecules are treated with the CG beads. This simulation is used to

study the structural and dynamical properties of G-protein coupled receptors, ABC receptors and insertion of a protein into a membrane [48–50].

3. **QM/MM/CG simulations**

   This hybrid simulation method involves all levels of detail, a specific portion of the system is treated with quantum details. Surrounding of the quantum representation is treated with the all-atom representation and the rest of the system is treated with the CG representation. [51].

4. **Adaptive resolution simulation**

   This hybrid simulation method allows to change the levels of detail of a molecule. Based on the predefined simulation region or based on some factor, a particle can adapt its representation from CG to AA to quantum mechanics or vice-versa [52–54].

1.1 Molecular Simulation

Since quantum mechanics methods involve electronic degrees of freedom, most of the problems are too large to be studied. Methods that ignore electrons and represent atoms as a nuclei, AA and CG, are often used to simulate complex and large systems. Molecular simulation of a system can be perform by using either MC or MD. In particular, MD is a method that involves integrating the Newton's equation of motions. In fact, the time evolution of a set of N particles can be derived from the Hamiltonian function:

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p} + V(\mathbf{q}) \tag{1.1}$$

$\mathbf{p},\mathbf{q}$ are the momenta and the position vectors. The first term in the Hamiltonian is the total kinetic energy and second term represents the potential energy of a system (which depends upon positions of particles). $\mathbf{M}^{-1}$ is an inverse mass matrix. The equation of motions associated with the Hamiltonian is given by:

$$\begin{aligned} \dot{q}(t) &= \nabla_P H(\mathbf{q},\mathbf{p}) = \mathbf{M^{-1}p} \\ \dot{p}(t) &= -\nabla_q H(\mathbf{q},\mathbf{p}) = -\nabla V(\mathbf{q}), \end{aligned} \tag{1.2}$$

$\dot{\mathbf{q}}(\mathbf{t})$ represents the velocity vector and $\dot{\mathbf{p}}(\mathbf{t})$ is an acting force on particles. Forces in the system arise due to the inter-particle interactions and most often these forces are derived from a potential equation (or force fields). In subsequent sections, we briefly discuss about formulation of a force field and algorithms to compute forces in a system.

1.1.1 Force Fields

A force field is a set of equations and associated parameters. These equations, based on atomic coordinates (positions of atoms), compute the potential energy of a system. In AA and CG representations, particles interact with each other using an empirical force field. The general functional form of a force field is given by equation 1.3.

$$\begin{aligned} U_{FF} &= U_{bonded} + U_{non-bonded} \\ U_{Bonded} &= U_{bond} + U_{angle} + U_{dihedral} \\ U_{non-bonded} &= U_{shortrange} + U_{longrange} \end{aligned} \tag{1.3}$$

Different force fields might have the same functional form and have different values for the associated parameters. Moreover, different force fields with the same functional form and different values of constant parameters, and force fields with different functional form, may produce comparable trajectories.

In a force field, interactions in a system are divided into bonded and non-bonded interactions.

Bonded interactions

Bonded interactions include interactions due to bonds, angles and dihedral angles. Bond and angle terms are computed using the Hooke's law or the harmonic potential. This is characterized by a force constant and an equilibrium value, Whereas a dihedral potential is evaluated using a cosine series (equation 1.4).

$$\begin{aligned} U_{bonded} &= K_b(r - r_0)^2 \\ U_{angle} &= K_\theta(\theta - \theta_0)^2 \\ U_{dihedral} &= K_\psi((1 + cos(n\psi - \delta)) \end{aligned} \tag{1.4}$$

K_b, K_θ, K_ψ are force constants and r_0, θ_0 are equilibrium values of angles and dihedral angles. These constants are derived from either experiments or quantum simulations.

Some biological force fields also include a improper dihedral term in the bonded term that include having a out-of-plane atom.

Non-bonded interactions

Non-bonded interactions in a biomolecular system are characterized by the interactions between particles that are not connected by a bond or an angle (sometime dihedral). In a biomolecular system non-bonded interactions are categorized as:

1. Lennard-Jones interactions
2. Electrostatic interactions.

Van der Waals interactions

In a force field, Lennard-Jones (LJ) interactions are frequently computed using the Lennard-Jones potential.

$$U_{LJ} = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{1.5}$$

U_{LJ} is the Lennard-Jones potential, also referred as the 12,6 or the LJ potential. In the LJ potential, ε is the depth of potential well, and σ is the inter-particle distance at which the potential is zero. For a pair $i-j$, the parameters ε and σ in the U_{LJ} are combined with mixing rules, which are as follows:

Geometric mean for $\varepsilon_{ij} = \sqrt{\varepsilon_i \varepsilon_j}$ and arithmetic mean for $\sigma = (\sigma_i + \sigma_j)/2$.

Electrostatic interactions

Charged particles in a system interact with each other through electrostatic interactions. Moreover, a molecule can interact with other molecules through electrostatic interactions if charges on molecules are distributed unevenly. For example two water molecules interact via electrostatic interactions because of electro-negativity difference of the H atom and the O atom or uneven charge distribution on the H-O-H molecule. The electrostatic interaction for an ionic pair i, j is computed using the Coulomb potential:

$$U_{coul} = K_{coul} \frac{q_i q_j}{r_{ij}} \tag{1.6}$$

In equation 1.6, parameter K_{coul} is a constant, q_i, q_j are charges on particles i and j respectively and r_{ij} is the distance between particles. Unlike the van der Waals potential, the Coulomb potential decays slowly with the distance that makes computationally expensive to evaluate.

1.1.2 Force fields for materials

The force field in the equation 1.3 (contain pair potentials) is generally used in performing simulation of materials. Over the past decade, several studies have reported the failure of pair potential to reproduce properties for materials like melting temperature and others [55, 56]. Therefore, material scientists have added an extra multi-body potential term in the force field formulation. The functional form of a force field for materials is given by the equation 1.7.

$$U_{total} = U_{AB}(r_{AB}) + U_{BC}(r_{BC}) + U_{AC}(r_{AC}) + U_{ABC}(r_{AB}, r_{BC}, r_{AC}) \tag{1.7}$$

The three-body term in equation 1.7 $U_{ABC}(r_{AB}, r_{BC}, r_{AC})$ can be positive or negative, depending upon the type of interaction. The Stillinger-Weber and the Tersoff potentials are examples of the three-body potential that are widely used in designing and simulation of semiconductors [56, 57]. The potential form of three particles A,B and C interacting with the Stillinger-Weber potential is given by:

$$U_{SW} = \frac{1}{2}\sum_{ij}\phi(r_{ij}) + \sum_{ijk} g(r_{ij})g(r_{ik})\left(cos\theta_{jik} + \frac{1}{3}\right)^2 \tag{1.8}$$

where $\phi(r_{ij}), g(r_{ij}), g(r_{ik})$ are two-body terms and θ_{jik} is a three-body angle term. Most of these force fields (pair potentials and three-body potential) depend mainly upon the inter-particle distances.

1.1.3 Reactive Force Field

A reactive force field is a bond order based force field that allows to stimulate the formation and breaking of bonds [58]. The bond order is computed from interatomic distances. The force field is mainly used in simulation of materials and hydrocarbons. ReaxFF is a reactive force field and the potential energy function of ReaxFF is given as:

$$U_{system} = U_{bond} + U_{over} + U_{angle} + U_{tors} + U_{vdWaals} + U_{Coulomb} + U_{Specific} \tag{1.9}$$

The U_{bond} term in eq 1.9 represents the potential energy associated with both a bond and formation of a bond between two atoms. Potential energies for a angle and a torsional angle are given by U_{angle} and U_{tors}. U_{over} is the energy penalty for an inappropriate bond (for example, there will a penalty, if the number of bonds form by an atom exceeds by the number of valance electrons). $U_{Coulomb}$ and $U_{vdWaals}$ terms are non-bonded interactions (similar to equations 1.5 and 1.6). $U_{Specific}$ is a system specific term that involves in capturing a specific conformation of a chemical compound.

1.2 Algorithms for force computation

In computer simulations of a system with all-atom (AA) or CG representation, force computation is the most time-consuming step. In fact, bonded terms are computed efficiently and the time-complexity of these terms is proportional to the number of particles present in the system. On the contrary, computation of non-bonded terms for a system containing N number of particles has the time-complexity of order N^2, if computed between every pair of particles. However, advanced algorithms are frequently used to reduce the time-complexity of non-bonded terms. In this section, we briefly discuss about algorithms to compute non-bonded forces efficiently.

Non-bonded interactions can be divided into two classes; short- and long- range interactions.

1.2.1 Short-range interactions

Short-range interactions are typically truncated (*i.e.* made to vanish after a specific *cutoff distance* r_c), and computed thanks to *neighbor lists*, *i.e.* lists of neighboring particles. These neighbor lists are often built through a combination of *cell lists* and *Verlet lists*[59]. In the cell lists method, all particles are binned into smaller $3D$ cells of side length $l \geq r_c$ according to their coordinates. This binning ensures that, for each particle, all interacting particles are either in the primary box or in the neighboring 26 cells (instead of the whole simulation box). Under the assumption that particles are distributed uniformly and the force calculation on a particle is proportional to the volume of neighbor region then the cell list method, for each particle, reduces the search volume from L^3 to $27r_c^3$ (L is the simulation box length).

In the Verlet lists method, each particle is associated to a list of neighboring particles within a distance $r_s = r_c + \delta$, where δ is a *buffering distance*. This method further reduces the search volume from $27r_c^3$ to $4\pi(r_c+\delta)^3/3$. However, a combination of cell and Verlet lists is often use in construction of neighbor lists. The combined method constructs neighbor list for

all particles by searching in $27r_c^3$ volume; however, neighbor lists contain all pairs that have a smaller distance than $(r_c + \delta)$. This combined approach allows to use same neighbor lists for few time steps and these lists are updated either every N time steps, or based upon how far the particles have moved [60, 61]. Algorithm 1 shows the steps to construct a neighbor list of a particle i, and $Cells(i)$ contains all indices of parrticles belonging to neighboring cells.

Algorithm 1: $NL(i)$

```
for j ∈ Cells(i) do
    r^2 = ||r(i) − r(j)||^2
    if r^2 ≤ (r_c + δ)^2 then
        NL(i) ← NL(i) ∪ j
    end
end
```

1.2.2 Long-range interactions

Long-range interactions are infinite ranged and calculations of these forces are computationally expensive. Unlike short-ranged interactions, long-rang interactions involve computation in a simulation box and all of its periodic images. In order to compute forces over all the periodic images, the Coulombic interaction for a set of point charges is given by:

$$U_{coul} = \frac{1}{4\pi\varepsilon_0} \sum_{\mathbf{n}} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ i \neq j}}^{N} \left(\frac{q_i q_j}{||\mathbf{r}_{ij} + \mathbf{n}||} \right) \tag{1.10}$$

where $\mathbf{n}$ are the lattice vectors $\mathbf{n} = (n_x L_x, n_y L_y, n_z L_z)$.
Methods to compute electrostatic interactions are broadly categorized into two categories:

- Ewald summation based methods: Particle Mesh Ewald , PPPM
- Non-Ewald summation based methods: Reaction field, Wolf method and Cut-off distance based methods.

Ewald Sum

Long-range interactions are often computed by using the Ewald summation method [62]. In the beginning, the Ewald summation method is used in computing electrostatic energies of ionic crystals, and later this method is used in molecular simulation to compute the long-range interactions (electrostatic forces). In Ewald summation approach, each point

charge in a system is surround by a counter-charge distribution i.e. a charge distribution of equal magnitude but opposite sign (a Gaussian distribution is commonly used). This method involves representing r^{-1} into the summation of two absolute and rapidly convergent series (eq. 1.11).

$$\begin{aligned} \frac{1}{r} &= \Phi_{real}(r) + \Phi_{recip}(r) \\ \Phi_{recip}(r) &= \frac{erf(r)}{r} \\ \Phi_{real}(r) = \frac{1}{r} - \frac{erf(r)}{r} &= \frac{erfc(br)}{r} \end{aligned} \tag{1.11}$$

This splitting involves a short-range $\Phi_{real}(r)$ and a long-range term $\Phi_{recip}(r)$. The short-range part is evaluated in real space by using the neighbor list algorithm (algo. 1) , whereas the long-range part $\Phi_{recip}(r)$ is computed using Fourier series in reciprocal space. erf and $erfc = 1 - erf$ in equation 1.11 are the error and the complementary error functions, respectively.

The Ewald summation scales as $\mathcal{O}(N^{\frac{3}{2}})$ (assuming a homogeneous charge distribution in the simulation box), and it can be computationally expensive for a very large system. Thus several alternative methods have been proposed to reduce the computational complexity of the reciprocal part of the equation 1.11. One alternative is the Particle-Particle-Particle-Mesh method (PPPM), which scales as $\mathcal{O}(N \ln N)$. In the PPPM method, charges are interpolated onto a mesh, and a gridded charge density is generated. The FFT algorithm is used on the gridded charge density to compute the reciprocal part of a electrostatic potential [63].

Another alternative method is the Fast Multipole Methods (FMM) [64]. In the FMM algorithm, a system is divided into cells, and based on a multipole expansion, charges are distributed in each cell. Finally, the potential is evaluated by computing interactions among these cells. The FMM algorithm scales as $\mathcal{O}(N)$.

1.3 Parallel algorithms for Molecular Dynamics

Molecular dynamics is frequently used to simulate large-scale problems. Serial execution of MD codes take a huge amount of time. On the contrary, parallel execution of these codes reduces the significant amount of computational time. All modern and popular MD simulators use parallization to take advantage of modern computer architecture and clusters. Besides the parallel MD codes, special-purpose super computers, Anton [65] and MD-grape [66], have

been designed to explore the hardware level performance of frequently used mathematical operations in molecular dynamics (specially square root operation) [67]. However, building a special-purpose super computer requires a lot of investment and dedicated software that is not frequently portable on other architectures [68]. The advent of modern architectures such as GPUs and Xenon-phi have pushed the parallelization capabilities and simulation length for molecular dynamics. These architectures provide enough computational resources to simulate complex and large-scale systems. However, new algorithms and data structures are require that can efficiently utilize these hardware resources.

In MD, computation of the total force acting on a particle is independent of forces on other particles. Due to this, MD has no data-dependency in the force computation routine and forces on all particles can be computed in parallel. In particular, parallelization of non-bonded interactions alone can significantly speed-up the simulation, as these interactions are computationally expensive and highly parallelizable. Most often, non-bonded interactions are computed using neighbor lists and these non-bonded interactions can be parallelized by providing almost equal number of neighbor lists to each thread. This approach is often used to simulate systems on a shared memory architecture using openMP and CUDA; however, on a distributed memory, computation of force acting on a particle requires position of all neighboring particles on the same processor. Parallelization on a shared memory architecture mainly involve distribution of force computations on different threads, other routines like position and momenta updates are also parallelized but overall computation time is dominated by force calculations. Depending on the hardware architecture, Newton's third law can be applied to further optimization of force calculations. On the GPU, Newton's third law requires random memory accesses for performing a read and a write operations, that poses a substantial performance penalty. The general steps to parallelize a MD code on a shared memory architecture are as follows:

1. Barrier
2. Compute forces on all particles over N threads
3. Barrier
4. Parallel update of momenta and positions

The first barrier ensures that force computations perform after all of the positions are updated and the second barrier enforces to compute forces on all the particles before position and momenta update. Since, momenta and position of a particle depend upon the force acting on it, both steps can be performed simultaneously and these steps do not require any

barrier. MD simulators, for the GPU, generally utilize the capabilities of both a CPU and a GPU. In fact, computation of less time consuming part such as momenta, position updates and bonded-force computations are performed on a CPU whereas non-bonded forces on all particles are computed on a GPU. This method is used in most software packages; however, this method requires transfer of positions of particles from a CPU to a GPU and transfer of computed forces from a GPU to a CPU. The transfer of data from a CPU (host) to a GPU (device) or vice-versa slows down the performance. Most often, these transfer routines are pipeline with CPU routines. Other methods involve performing a complete simulation on either GPUs or CPUs.

In MD, a system containing N particles can be divided on P processes based on any of these three algorithms.

1. Atom-Decomposition Algorithm: In this method, N particles are divided over P processors into a set of N/P particles. Each processor is responsible for computing the forces on, and updates the positions and momenta of, its N/P particles. The major drawback of this method is that this method requires two global communication operations at each integration step. The first communication assigns the updated positions of all particles on all the processors, and second communication ensures that every processor receives the force on its particles computed by other processors. Biological MD simulators (CHARMM [69] and GROMOS [70]) use this algorithm for distributing particles over the processors.

2. Force-decomposition Algorithm: This method distributes a subset of the force matrix over the processors. This method is not used in any popular MD simulators.

3. Spatial-Decomposition Algorithm: In a spatial decomposition algorithm, a simulation box is subdivided P smaller sub-boxes or domains such that all processes have one box. A processor is responsible for computing forces and updating positions and momenta of its own particles. Additionally, each processor is also responsible for sending and receiving particles from its neighboring processors. During the simulation, particles can migrate from one sub-domain to another sub-domain. Due to these migrations, particles need to be redistributed after certain time steps over all processors.

   Algorithms have been designed to take advantage of Newton's third law by sending positions of particles in the forward directions and receiving forces in backward directions. This technique is most effective if the distribution of particles in the system is uniform. This method is commonly used in modern simulator and biological MD programs such as AMBER [71], GROMACS [72] and NAMD [73] and LAMMPS [74].

The spatial decomposition algorithm is best suited for simple and homogeneous systems. This algorithm might suffer from a load imbalance and gave a poor scaling for complex and non-uniform systems such as biological systems.

1.4 Simulating a system

Integrating the equation of motion yields a trajectory that contains position and velocities of each particle present in the system. This trajectory is then used to compute statistical properties of the system which are functions of positions or velocities or functions of both positions and velocities. Properties that depend upon position are referred as structural properties of the system like, a radial distribution function, radius of gyration, root mean square deviation and solvent accessible surface area etc. Averages or properties based upon velocities or momenta (or both) are called dynamical properties of the system. Velocity-auto-correlations and computing diffusion constants are some examples of dynamical properties.

Generally, in MD, the next state of the system y_{t+1} is derived from the current state y_t. At each step, forces on each particles are computed, and then these forces are used with current positions and momenta to generate the positions and momenta of the next state. This process is repeated for a desired number of steps. A MD simulation generates a trajectory that contain information about positions and momenta as a function of time.

Solving the Hamiltonian equation of a system allows one to explore the constant energy surface of the system. However, some experiments obtained the statistical properties by fixing either temperature or pressure or both constant. In statistical mechanics, the experimental average of an observable is defined with the ensemble averages. The ensemble averages are averages obtained with a large number of replicas of the same system at a same time. A system is simulated using one of these ensembles:

Microcanonical ensemble (NVE) : This ensemble is a constant-energy ensemble that keeps number of particles N, volume of the system V, and total energy of the system E, constant. A simulation in this ensemble does not use any temperature or pressure control.

Canonical Ensemble (NVT): In this ensemble, a system evolves with a constant number of particles N, a constant volume V, and a constant temperature T. This ensemble is frequently used to simulate biological problems such as protein folding.

Isobaric-Isothermal Ensemble (NPT): This ensemble keeps a constant number of particles N, a constant pressure P, and a constant temperature T. This ensemble allows the simulation

volume to expand.

Grand canonical Ensemble (μVT): This ensemble is keeps a fixed chemical potential μ, a fixed volume V, and a fixed temperature T.

These ensembles are used for simulating a system, and then obtaining the statistical average of an observable. In general, the value of an observable depends upon the positions and the momenta of particles present in the system, thus, an observable A can be represents as $A(\mathbf{p},\mathbf{q})$. The experimental value of the observable corresponds to the average of the observable over time. One way to obtain these averages is to simulate the dynamical behaviour of the system. At the end, averages of a desired property can be obtained by:

$$< A >= \frac{1}{N_{config}} \sum_{i=1}^{N_{config}} A(r^N) \tag{1.12}$$

These obtained statistical averages are subject to two types of errors: systematic bias and statistical errors. Systematic bias is mainly ue to the use of a discrete time step (and other optimization techinques), and statistical errors occur due to the quality of sampling. The statistical error in averages may be estimated by computing the variance (σ^2) of an observable (given by eq. 1.13). The variance can be measured either by correlation time analysis or by a block-averaging scheme [75–78].

$$\sigma_A^2 =< (A- < A >)^2 > \tag{1.13}$$

The standard deviation, square root of the variance, can be computed by using the equation 1.14. The standard deviation of an estimated average is inversely proportional to the number of generated configurations. Hence, a longer simulation will reduce the error in averages and a more precise value of averages can be computed.

$$\sigma_{<A>} = \frac{\sigma_A}{N_{config}} \tag{1.14}$$

1.5 Adaptively Restrained Molecular Dynamics

In MD, particles update their positions at each time-step. These updates of positions lead to changes in the inter-particle distances, hence, forces need to be updated. In Adaptively restrained molecular dynamics (ARMD), particles update their positions based on their kinetic energy. Particles with kinetic energy smaller than a threshold are considered as

restrained or frozen at a given time-step. Particles with kinetic energy greater than the threshold are either in transition region or in normal dynamics region and these particles are *active* particles. Since inter-particle distances between frozen particles do not change, forces between them also do not change. The forces between frozen particles do not need to be computed at each time step and forces involving active particles need to be computed at each time-step.

In MD, The time evolution of the system containing N particles may be derived from the Hamiltonian function 1.15

$$H(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p} + V(\mathbf{q}) \tag{1.15}$$

where $\mathbf{q}$ is a $3N$-dimensional vector of coordinates, $\mathbf{p}$ is a $3N$-dimensional vector of momenta, $\mathbf{M}$ is a $3N \times 3N$ mass matrix, and $V(\mathbf{q})$ is the potential energy.

In ARMD, the $3N \times 3N$ inverse mass matrix $\mathbf{M}^{-1}$ is replaced by a $3N \times 3N$ inverse inertia matrix $\Phi(q,p)$ which adaptively enforces restraints during simulation [79]:

$$H_{AR}(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\Phi(\mathbf{q},\mathbf{p})\mathbf{p} + V(\mathbf{q}). \tag{1.16}$$

One possible choice for the inverse inertia matrix $\Phi(\mathbf{q},\mathbf{p})$ is a block-diagonal matrix $diag\,[\Phi_1(\mathbf{q}_1,\mathbf{p}_1),\ldots,\Phi_N(\mathbf{q}_N,\mathbf{p}_N)]$ with

$$\Phi_i(\mathbf{q}_i,\mathbf{p}_i) = m_i^{-1}[1-\rho_i(\mathbf{p}_i)]\mathbf{I}_{3\times 3} \tag{1.17}$$

where $\mathbf{I}_{3\times 3}$ is the 3×3 identity matrix, m_i, $\mathbf{q}_i$ and $\mathbf{p}_i$ are respectively the mass, position and momentum of particle i, and ρ_i is its *restraining function*:

$$\rho_i(\mathbf{p}_i) = \begin{cases} 1, & \text{if } 0 \le K_i(\mathbf{p}_i) \le \varepsilon_i^r \\ 0, & \text{if } K_i(\mathbf{p}_i) \ge \varepsilon_i^f \\ s(K_i(\mathbf{p}_i)) \in [0,1] & \text{elsewhere} \end{cases} \tag{1.18}$$

In this definition, $K_i(\mathbf{p}_i) = \frac{1}{2}m_i^{-1}\mathbf{p}_i^T\mathbf{p}_i$ is the kinetic energy of particle i, ε_i^r is its *restrained-dynamics threshold*, ε_i^f is its *full-dynamics threshold*, and s is a $\mathscr{C}^2$ function that smoothly interpolates between 0 and 1.

When $\rho_i = 0$, $\Phi_i = m_i^{-1}$ and particle i is *active* (the particle mass is unchanged). When $\rho_i = 1$, $\Phi_i = 0$ and particle i is *restrained*, *i.e.* will not move whichever force is applied to it (the particle mass is infinite). When $\rho_i \in (0,1)$, particle i is in a *transition state*. The restraining function ρ_i above depends upon the kinetic energy of the particle. Precisely,

particle i is restrained when its kinetic energy is smaller than the restrained-dynamics threshold (ε_i^r), and a particle will have normal dynamics if its kinetic energy is larger than its full-dynamics threshold ε_i^f. Particles with kinetic energies between ε_i^r and ε_i^f are considered as transition particles.

To integrate the equations of motion of a system in the NVE ensemble using the AR Hamiltonian, we may use a modified Velocity Verlet algorithm which takes into account the non-constant mass matrix:

1. $\mathbf{p}_i(t+\frac{1}{2}\Delta t) = \mathbf{p}_i(t) + \frac{1}{2}\mathbf{f}_i(t)\Delta t$
2. $\mathbf{q}_i(t+\Delta t) = \mathbf{q}_i(t) + \nabla_{\mathbf{p}_i} H_{AR}(t+\frac{1}{2}\Delta t)\Delta t$
3. $\mathbf{f}_i(t+\Delta t) = -\nabla_{\mathbf{q}_i} V(\mathbf{q}(t+\Delta t))$
4. $\mathbf{p}_i(t+\Delta t) = \mathbf{p}_i(t+\frac{1}{2}\Delta t) + \frac{1}{2}\mathbf{f}_i(t+\Delta t)\Delta t$

To perform simulation in the canonical ensemble (NVT), we use AR Langevin dynamics [79, 80]:

$$\begin{aligned} d\mathbf{q} &= \nabla_{\mathbf{p}} H_{AR}(\mathbf{q},\mathbf{p})dt \\ d\mathbf{p} &= -\nabla_{\mathbf{q}} H_{AR}(\mathbf{q},\mathbf{p})dt - \gamma\nabla_{\mathbf{p}} H_{AR}(\mathbf{q},\mathbf{p})dt + \sqrt{\frac{2\gamma}{\beta}}d\mathbf{W} \end{aligned} \tag{1.19}$$

where dW_t is a $3N$-dimensional Brownian motion and $\gamma > 0$ is the frictional constant. Discretization of the modified Langevin is done using second-order Trotter splitting [80]. The temperature of the AR system is given by:

$$T = \frac{1}{DK_B}\left\langle \sum_{i=1}^{N}\left(\mathbf{p}_i \cdot \frac{\partial H_{AR}}{\partial \mathbf{p}_i}\right)\right\rangle \tag{1.20}$$

where D is the number of degrees of freedom in the system and the dot represents the dot product. In the version described above, the AR Hamiltonian is separable and the temperature is unchanged [79].

During the simulation particles might change their state, and the Adaptively Restrained Hamiltonian ensures that stable simulations can be performed. Equilibrium statistics can be recovered by performing the NVT simulation using the AR Langevin dynamics.

ARMD accelerates the simulation speed by reducing the number of force computations associated with restrained-restrained particles at each integral time-step. In order to speed up the simulation using ARMD, new algorithms in conjugation with classical neighbor list

algorithms are require. The new algorithms should not only use neighbor lists and Newton's third law but also neglect the computation of restrained-restrained forces.

1.6 Contributions

The main goal of this thesis was to develop new algorithms for adaptively restrained molecular dynamics. Precisely, we developed algorithms for neighbor list and force computations. The main advantage of these algorithms over MD simulations is the reduction in the number of force computations at each time step, which is the most time-consuming part in MD simulations. We designed the active neighbor list (ANL) algorithm, that stores all the pairs that involve at least one active particle. These ANLs are updated incrementally for the particles that switch from active to restrained state or vice-versa.

The developed algorithms were implemented for constructing ANLs in the LAMMPS MD simulator, which has been published in the *Modelling and Simulation in Materials Science and Engineering* journal [81]. However, these algorithms can be applied to other MD packages like GROMACS, CHARMM and NAMD. We have also proposed a method to implement new methodology in LAMMPS, without the need to modify the LAMMPS source code. These proposed algorithms and methods, along with their validation on different toy model systems are described in Chapter 2.

The force computation algorithm in Chapter 2 uses a two-pass incremental force update that involves force increments to be calculated twice, which limits the speed up of ARMD. Chapter 3 presents a single-pass incremental force update algorithm, which overcomes the aforementioned limitation and improves the speed and performance of ARMD. The new proposed algorithm has been validated on diverse force fields and multiple systems. We demonstrated that these algorithms, in conjugation with the Wolf method, can also be used for the computation of electrostatic forces. Our results show, up to one order of magnitude speed up using ARMD as compared to MD. In addition, for a polymer in solvent in the canonical ensemble, we have shown that ARMD not only reproduces the statistical averages obtained by conventional MD, but also speeds up the convergence of an observable in wall-clock time. This work has been published in the Journal of Computational Chemistry.

Chapter 4 introduces parallel algorithms for single-pass incremental force computations using active neighbor lists, to take advantage of adaptive restraints using the Message Passage Interface (MPI) standard. This work has been accepted for publication in the 2017 *International Conference on High Performance Computing and Simulation (HPCS 2017)*.

Chapter 2

Adaptively Restrained Molecular Dynamics in LAMMPS

Abstract

Adaptively restrained molecular dynamics (ARMD) is a recently introduced particles simulation method that switches positional degrees of freedom on and off during simulation in order to speed up calculations. In the NVE ensemble, ARMD allows users to trade between precision and speed while, in the NVT ensemble, it makes it possible to compute statistical averages faster. Despite the conceptual simplicity of the approach, however, integrating it in existing molecular dynamics packages is non-trivial, in particular since implemented potentials should a priori be rewritten to take advantage of frozen particles and achieve a speed-up. In this chapter, we present novel algorithms for integrating ARMD in LAMMPS, a popular multi-purpose molecular simulation package. In particular, we demonstrate how to enable ARMD in LAMMPS without having to re-implement all available force fields. The proposed algorithms are assessed on four different benchmarks, and show how they allow us to speed up simulations up to one order of magnitude.

2.1 Introduction

Adaptively Restrained Molecular Dynamics (ARMD) is a recent approach that attempts to tackle the timescale issue by reducing the number of computations per time step [79]. In ARMD, particles adaptively switch their positional degrees of freedom on and off during the simulation, based on their instantaneous kinetic energy. Precisely, particles whose kinetic energy is sufficiently large are considered *active*, and have normal dynamics, while particles whose kinetic energy is below some threshold are *restrained*, and stop moving completely. The status of particles evolve during simulation, and the Adaptively Restrained Hamiltonian ensures that stable simulations can be performed, and that statistics can be recovered [79]. Since inter-atomic forces typically depend upon relative particle positions, only forces involving active particles need to be updated at each time step [82]. This may result in significant speed-ups, depending on the chosen simplification thresholds.

In this chapter, we introduce a novel method to compute neighbor lists and short-range forces when performing adaptively restrained molecular dynamics simulations. This method is independent of the underlying potential or force field, and may be used with any pair potential. We demonstrate our approach through an implementation in LAMMPS, a popular MD package [74], and validate it by simulating several systems in the NVE and NVT ensembles. We show that our algorithms, combined with the AR molecular dynamics methodology, makes it possible to finely trade between precision and computational cost (in the NVE ensemble), and speed up the calculation of statistical properties (in the NVT ensemble).

2.2 Methods

2.2.1 Molecular dynamics in LAMMPS

After setting up initial conditions, LAMMPS repeats the following steps:

Algorithm 2: LAMMPS integration step

```
1 if (UpdateNeeded) then
2     Build neighbor lists
3 end
4 Calculate forces
5 Update momenta
6 Update positions
```

where lines $1-2$ ensure that neighbor lists are rebuilt periodically (*e.g.* every 20 time steps), and line 4 is the main step: calculating forces.

To calculate forces, LAMMPS provides force fields implementations with a list L of particles for which forces should be computed, as well as the neighbor list $NL(i)$ of each particle i in L. LAMMPS may provide either *full neighbor lists* (FNLs) or *half neighbor lists* (HNLs). In the FNL case, all neighbors of particle i are stored in $NL(i)$. In the HNL case, the neighbor pair (i, j) is stored in either $NL(i)$ or $NL(j)$. The HNL case is used when the force calculation step may use Newton's third law ($\mathbf{f}_{ij} = -\mathbf{f}_{ji}$) in order to reduce the number of force calculations by half and speed up the simulation.

After initializing forces as zero, the force calculation algorithm is then:

Algorithm 3: $ComputeForces(L, NL)$

```
1 for (i ∈ L) do
2     for (j ∈ NL(i)) do
3         f_ij ← ComputeForce(i, j)
4         f_i ← f_i + f_ij
5         if (NewtonOn) then
6             f_j ← f_j − f_ij
7         end
8     end
9 end
```

where $\mathbf{f}_{ij}$ is the force applied to i by j, $\mathbf{f}_i$ is the total force applied to i, and $NewtonOn$ is true when HNLs and Newton's third law are used.

2.2.2 Adaptively Restrained Molecular Dynamics in LAMMPS

In AR molecular dynamics, a system of N particles can be represented as a combination of N_R restrained particles and $N_A = N - N_R$ active or transitioning particles. At any time-step, interactions between particles (*i.e.* inter-particle energies and forces) may thus be categorized into two types:

1. **Restrained interactions**: interactions between restrained particles only.
2. **Active interactions**: interactions involving at least one active particle.

In most force fields, interactions only depend on *relative particle positions*. As a result, at any time step, restrained interactions do not have to be updated, and only active interactions must be recalculated (since active particles may have moved since the previous time step).

An ARMD integration step may thus rely on the same general steps as in Algorithm 2, but efficient ARMD simulations require *incremental force calculation algorithms*, *i.e.* force calculation algorithms that only update active interactions.

In order to enable AR molecular dynamics in LAMMPS without modifying the source code of all force fields, our strategy is a) to pass modified information to force fields implementations, unbeknownst to them, so that they *only compute active interactions*, and b) to use these active interactions to incrementally update the total forces applied to particles (which are required to update positions and momenta). We note that modifying LAMMPS in such a way allows us to enable ARMD for force fields that are *not yet implemented*, which is a significant advantage of the approach proposed in this chapter.

2.2.3 Active neighbor lists

Let C denote the complete list of particles, A denote the list of active (or transitioning) particles, and R denote the list of restrained particles, so that $C = A \uplus R$.

Since we want force fields implementations to compute active interactions only, we are going to pass them the list A of active particles instead of the complete list C. Assuming we may use Newton's third law, the list A is sufficient, since any force $\mathbf{f}_{ij}$ we need to compute involves at least one active particle. However, we cannot just pass to force fields the HNLs of active particles, since there would be a risk that these HNLs do not contain all the neighbors for which we need to compute interactions[1]. Conversely, we should not pass the FNLs of active particles if we want to take advantage of Newton's third law. As a result, we introduce *active neighbor lists* (ANLs), *i.e.* neighbors lists that are built in such a way that, when i is an active particle, $ANL(i)$ contains j if a) j is restrained or b) if j is active and $ANL(j)$ does not contain i. If i is restrained, $ANL(i)$ is empty. The ANLs can thus be seen as HNLs for active-active neighbors (if both i and j are active, either $ANL(i)$ contains j or $ANL(j)$

[1]Consider *e.g.* the case of four particles $1,\ldots,4$ that are all neighbors to each other, where $HNL(1) = \{2,3,4\}$, $HNL(2) = \{3,4\}$, $HNL(3) = \{4\}$ and $HNL(4) = \emptyset$, and assume that particles 1 and 4 are active. If a force field implementation only receives $HNL(1)$ and $HNL(4)$, it will only compute active interactions $\mathbf{f}_{12} = -\mathbf{f}_{21}$, $\mathbf{f}_{13} = -\mathbf{f}_{31}$, and $\mathbf{f}_{14} = -\mathbf{f}_{41}$ (thanks to $HNL(1)$), and will compute neither $\mathbf{f}_{42} = -\mathbf{f}_{24}$ nor $\mathbf{f}_{43} = -\mathbf{f}_{34}$, since neither $HNL(1)$ nor $HNL(4)$ signal that particles 2 and 3 are neighbors of particle 4.

contains i), and FNLs for active-restrained neighbors (if i is active, $ANL(i)$ contains all the restrained neighbors of i). We may thus use the following algorithm to build the ANLs:

Algorithm 4: *BuildANL*(i)

```
for (j ∈ FNL(i)) do
    if (j ∈ R) or (i ∉ ANL(j)) then
        ANL(i) ← ANL(i) ∪ j
    end
end
```

When we initialize the simulation (before the first time step), we first build the ANLs from the FNLs thanks to Algorithm 4. During the main loop, however, we incrementally update the ANLs based on the list S_A of particles switching to an active (or transitioning) state, and the list S_R of particles switching to a restrained state. Precisely, if A and R represent the state distribution of the simulation at time step n, then S_A and S_R, ($S = S_A \bigcup S_R$), are the lists of particles switching between time steps n and $n+1$. We update the ANLs in two steps:

Step 1: clearing ANLs of particles that become restrained

When a particle i switches from active to restrained, we need to empty $ANL(i)$. However, we need to retain interactions with any neighboring particle j that remains active, *i.e.* insert i in $ANL(j)$. In the first step of the ANL lists update, we thus go through each particle i in S_R and clear $ANL(i)$ using Algorithm 5.

Algorithm 5: *ClearANL*(i)

```
for (j ∈ ANL(i)) do
    if (j ∈ A) and (j ∉ S) then
        ANL(j) ← ANL(j) ∪ i
    end
end
ANL(i) ← ∅
```

Step 2: building ANLs of particles that become active

When a particle i switches from restrained to active, we need to build $ANL(i)$. In the second step of the ANL lists update, we thus go through each particle i in S_A and build $ANL(i)$ using Algorithm 4.

2.2.4 Incremental force updates

As noted before, in order to achieve a speed-up through AR molecular dynamics, we need to be able to *incrementally* update the total forces applied on particles, *i.e.* avoid recomputing all total forces.

Let I denote the list of *involved particles*, *i.e.* the list of particles involved in active interactions (either because they are active particles, or because they are neighbors of active particles): $I = A \cup (\cup_{i \in A} ANL(i))$. The list I may be incrementally updated at each time step while updating the ANLs.

We may use A and the ANLs to incrementally update the total forces of particles in I:

1. Compute forces $\mathbf{f}^+_{old}$ between particles in I using current positions.
2. Subtract $\mathbf{f}^+_{old}$ from the total forces $\mathbf{f}$.
3. Update the positions of active particles.
4. Compute forces $\mathbf{f}^+_{new}$ between particles in I using the new positions.
5. Add $\mathbf{f}^+_{new}$ to the total forces $\mathbf{f}$.

Algorithm 6 shows the pseudo-code of the initialization step (before the first time step), and Algorithm 7 gives the pseudo-code for performing an AR molecular dynamics step in LAMMPS. This algorithm is illustrated in figure 2.1.

Algorithm 6: AR-LAMMPS initialization step

```
1  for (i ∈ L) do
2      f_i ← 0
3      f_i^+ ← 0
4      ANL(i) ← ∅
5  end
6  Construct all FNLs
7  f ← ComputeForces(C, FNL)
8  Build A, R, I
9  for (i ∈ A) do
10     BuildANL(i)
11 end
```

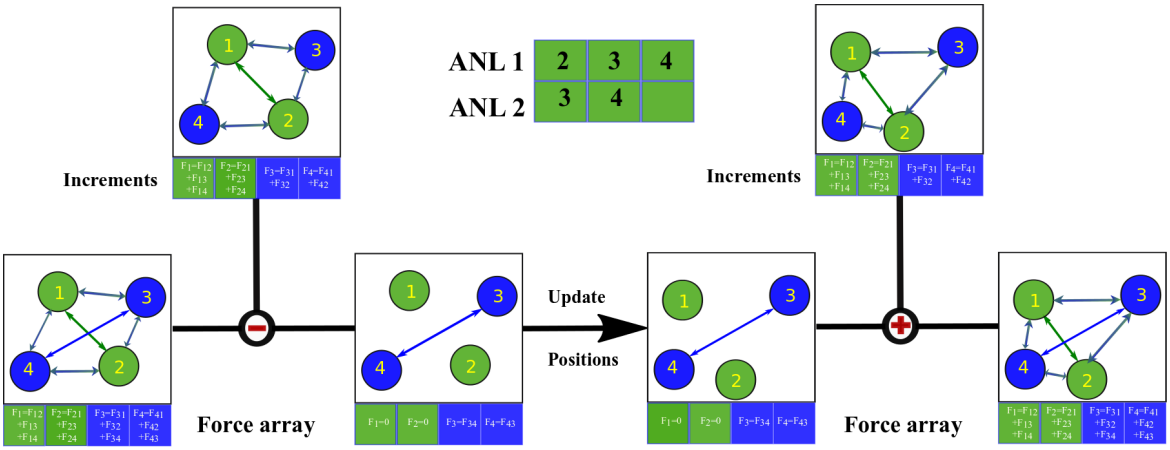


Fig. 2.1 A System of four particles, where all particles are neighbors to each other. Particles in green color represent active and in blue represent restrained particles (particle 1 and 2 are active while 3 and 4 are restrained). b) ANL of particle 1 and particle 2. Subtraction step: When compute function is called at first time step, forces are computed on all particles and stored in the force array. In order to remove the forces involving active particles, force increments using ANL are computed. These increments are then subtracted from the the force array. Subtraction steps followed by the update position step. In this step positions of the active particles are updated. After updating the positions, force increments based on new position are computed. In order to achieve total force on all the particles, force increments are added in the force array.

Algorithm 7: AR-LAMMPS integration step

Update momenta
Update A, R
if $(UpdateNeeded)$ **then**
    Construct all FNLs and empty all ANLs
    for $(i \in A)$ **do**
        $BuildANL(i)$
    end
end
else
    Build S_A and S_R
    for $(i \in S_R)$ **do**
        $ClearANL(i)$
    end
    for $(i \in S_A)$ **do**
        $BuildANL(i)$
    end
end
Update I
$\mathbf{f}^+ \leftarrow ComputeForces(A, ANL)$
for $i \in I$ **do**
    $\mathbf{f}_i \leftarrow \mathbf{f}_i - \mathbf{f}_i^+$
    $\mathbf{f}_i^+ \leftarrow 0$
end
Update positions
$\mathbf{f}^+ \leftarrow ComputeForces(A, ANL)$
for $i \in I$ **do**
    $\mathbf{f}_i \leftarrow \mathbf{f}_i + \mathbf{f}_i^+$
    $\mathbf{f}_i^+ \leftarrow 0$
end

2.3 Results and discussion

In order to validate our algorithms, we performed ARMD simulations on a set of toy models that were general enough to model typical simulations, while simple enough to allow for detailed analysis.

2.3.1 Systems of Lennard-Jones particles

We simulated three systems with different numbers (500, 4,000 and 108,000) of Lennard-Jones particles using an AR integrator in the NVE ensemble. All simulations were performed in reduced, Lennard-Jones units using our version of LAMMPS. Particles were generated along a fcc lattice of density 0.8442, and initial velocities were assigned according to the Boltzmann distribution with temperature as 1.2. For all simulations, we used a time-step of 0.005. Interactions beyond distance 2.5σ were ignored. Periodic boundary conditions were applied in all three directions. We chose values of ε_r and ε_f in order to achieve specific ratios of restrained particles.

We ran one reference simulation and one ARMD simulation for all three systems. Reference simulations were performed with the original LAMMPS neighbor list and force update algorithms, whereas ARMD used ANLs and incremental force update algorithms. To determine the resulting speed-ups, we compared the average times spent in each integration steps. Figure 2.2 shows the achieved speed-ups with respect to the percentage of restrained particles in the system. Figure 2.2 shows that, in order to have a speed-up for systems containing 500 and 4000 particles, at least 60% of the particles should be restrained. A 1.6X to 2.8X speed-up was observed when 80% to 90% particles were restrained. For the system containing 108,000 particles, we observed a speed-up when at least 50% of the particles were restrained, and a 2.5X to 4.2X speed-up was observed when 80% to 90% of the particles were restrained. One reason for achieving higher speed-ups in larger systems is the number of force calculations performed per time step. Smaller systems contain relatively fewer force calculations per time-step, and reducing them does not have much influence on the overall speed-up. For larger systems, force calculations constitute the major part of the time-step cost, and reducing them can significantly speed-up the simulation. Figure 2.3 shows that, even for AR simulations, the total adaptive energy of the system is constant[2].

In order to study the structural properties of the system, we also performed NVT simulations of all three systems using two different sets of AR parameters. We computed the radial distribution function (RDF) of the systems using classical MD and ARMD. Figure 2.4 shows that the RDFs obtained by both methods coincide.

2.3.2 Collision cascade

We performed this benchmark to show that AR molecular dynamics simulations in the NVE ensemble allow us to finely trade between precision and speed. In this benchmark, a

[2]Note that different AR parameters generate different Hamiltonians, so that identical initial conditions result in different total adaptive energies.

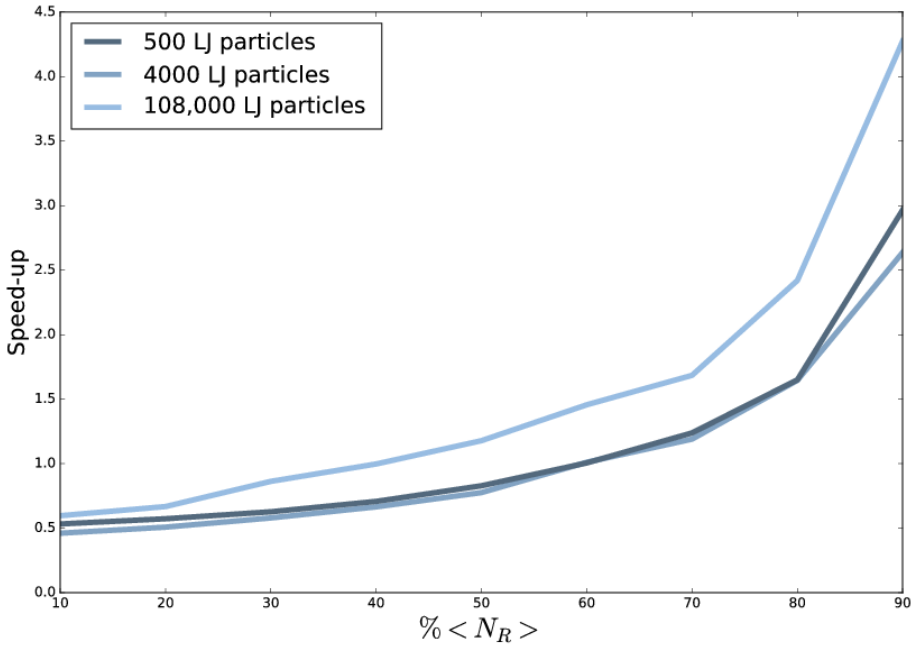


Fig. 2.2 Speed-up achieved with ARMD as a function of the percentage of restrained particles.

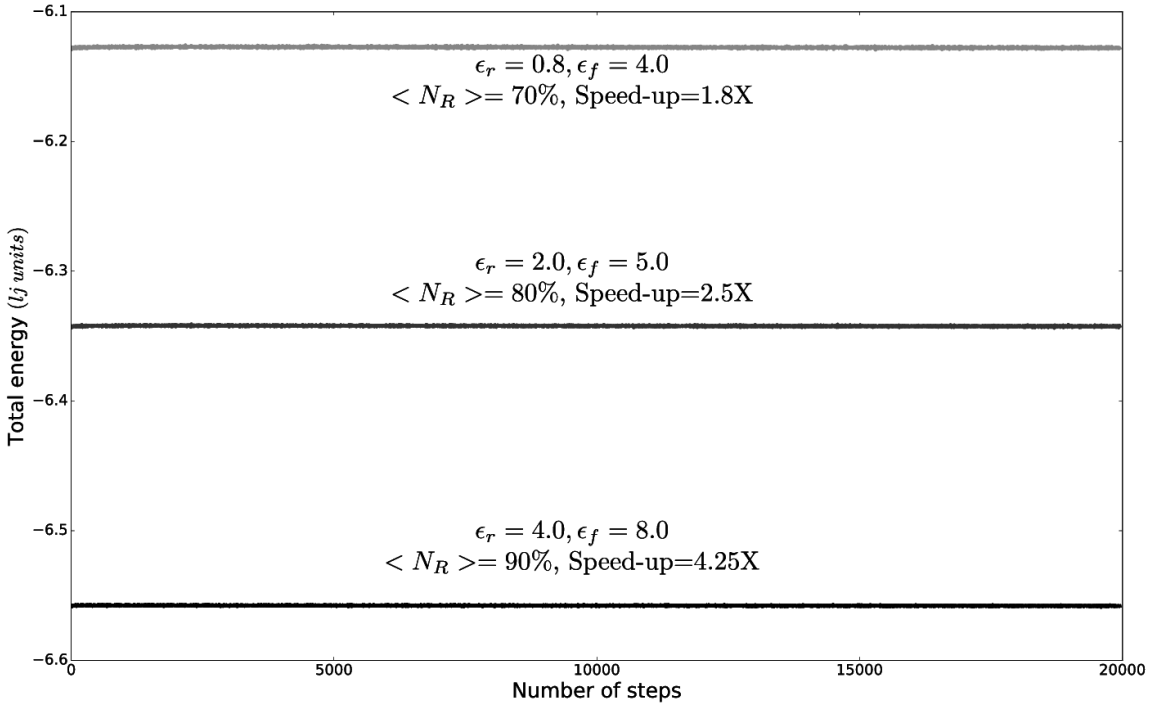


Fig. 2.3 ARMD simulation of 108,000 LJ particles with different AR parameters. The total energy remain constant irrespective of the AR parameters.

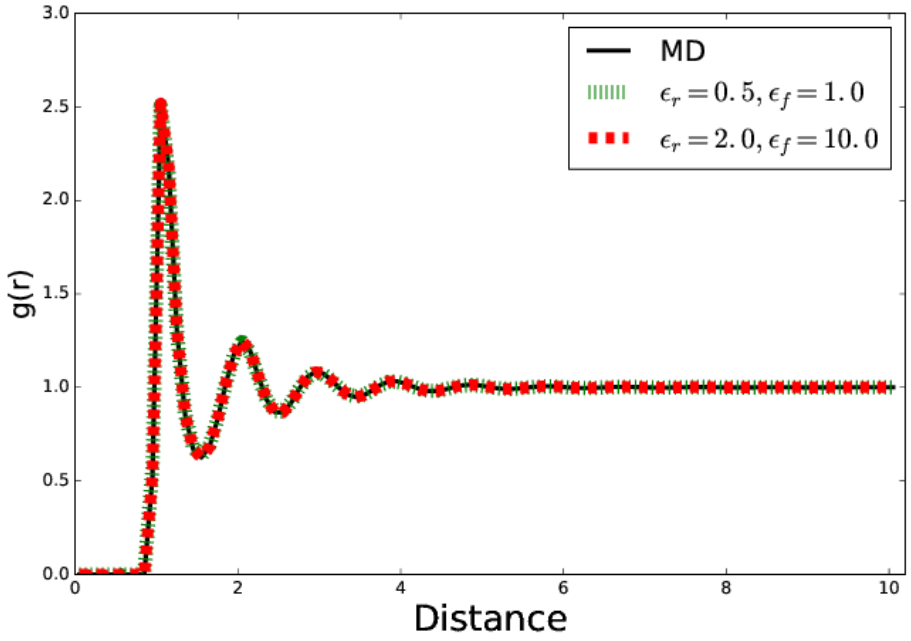


Fig. 2.4 AR molecular dynamics preserves the radial distribution function in the NVT ensemble (108,000 LJ particles).

high-velocity particle collides with an initially static 2D system containing 7290 particles. Interactions among particles are computed using a Lennard-Jones potential. We simulated this system for different values of AR parameters ε_r and ε_f, and various ratios $\frac{\varepsilon_f}{\varepsilon_r}$. All simulations were performed for 7000 steps with a time step size equal to 0.0003 (LJ units). We also performed a reference MD simulation of the same system using a Verlet integrator. To measure the deviation between ARMD simulations and the reference simulation, we extracted the last configuration of each ARMD simulation and computed the Root Mean Square Deviation (RMSD) with the last configuration of the reference simulation. In order to measure the speed-up achieved by AR simulations relatively to the classical simulation, we ran all simulations ten times and calculated the average computational cost of each time step.

Figure 2.5 illustrates the obtained speed-up as a function of AR parameters. We observed the highest speed-up (8.0 times) with $\varepsilon_f = 3 * \varepsilon_r$. As the ratio of $\varepsilon_f/\varepsilon_r$ increases, speed-up does not vary that much. For most values of AR parameters, we achieved 7 to 8 times speed-up. One reason for the smaller speed-up when the ratio between ε_f and ε_r increases is the larger number of particles belonging to the transition region, since updating positions of transitioning particles is more computationally involved than for particles belonging to other regions.

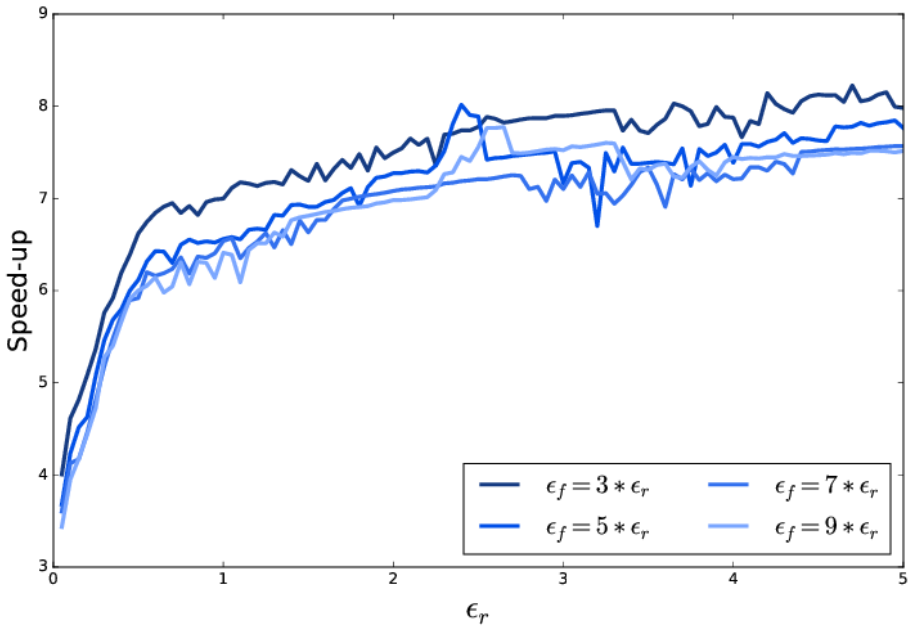


Fig. 2.5 Collision cascade: speed-up with respect to the ε_r values.

Figure 2.6 shows how ARMD deviates from standard MD when the AR parameters vary. From figure 2.6, we can infer that different AR parameters might result in the same RMSD, and that the relationship between the RMSD and the AR parameters may sometimes be non-trivial (such as the RMSD behavior when $\varepsilon_f = 3 * \varepsilon_r$). Figure 2.7 represents the speed-up as a function of the average percentage of restrained particles. The speed-up is highly correlated with the average number of restrained particles, and thus to values of ε_r. In this benchmark, we obtained an eight times speed-up when more than 98% of the particles were restrained, while still obtaining a RMSD from the reference simulation of 0.07 σ.

From this benchmark, it is evident that the achievable speed-up is related to the average number of restrained particles, which in turn is a function of ε_r. Higher ε_r and ε_f values lead to higher speed-ups and, in general, higher RMSD values.

2.3.3 Toy model of an ion passing through a membrane channel

In order to mimic the biological process of an ion passing through a membrane channel, we created a 2D toy model. In biology, channels are usually proteins that are immersed in biological membranes. These proteins either have a channel or pore, or act as a gate controlling the passage of small molecules across the membrane. The movement of the small molecules is often driven by an electrochemical gradient across the membrane. In this toy model, the membrane and ions were represented with Lennard-Jones particles, and we

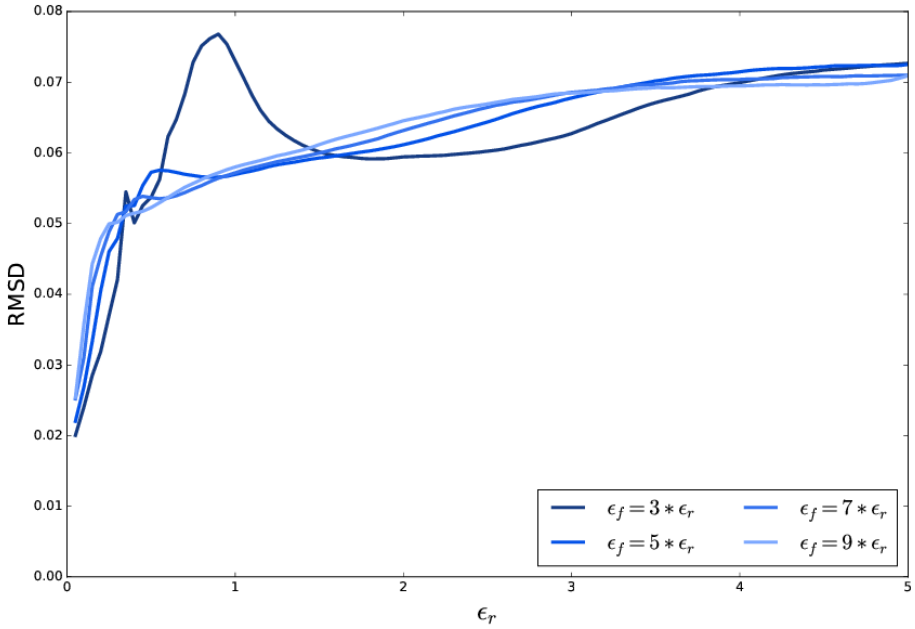


Fig. 2.6 Collision Cascade: deviation from standard molecular dynamics with respect to the ε_r values. Different colors indicate the relationship between ε_r and ε_f values.

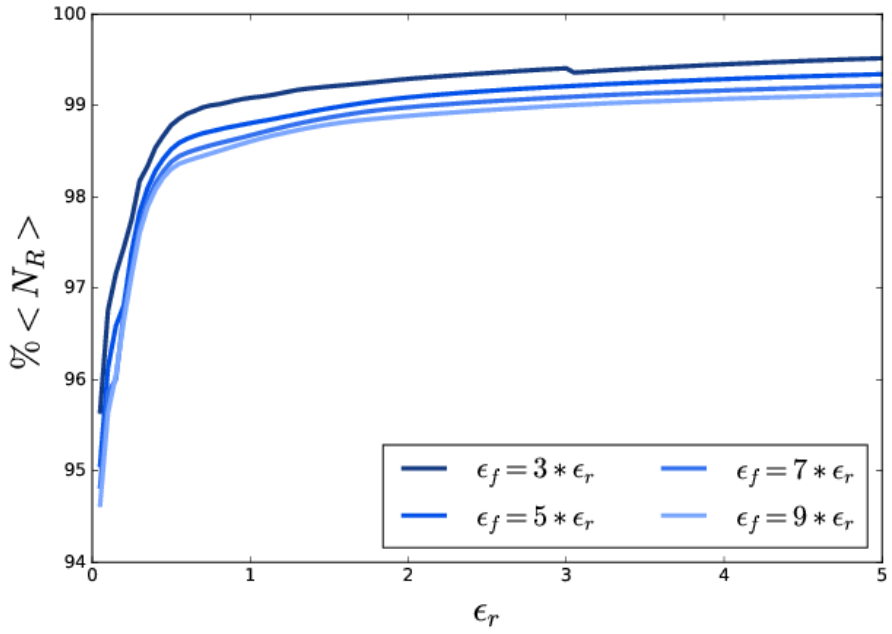


Fig. 2.7 Collision Cascade: number of restrained particles as a function of ε_r values.

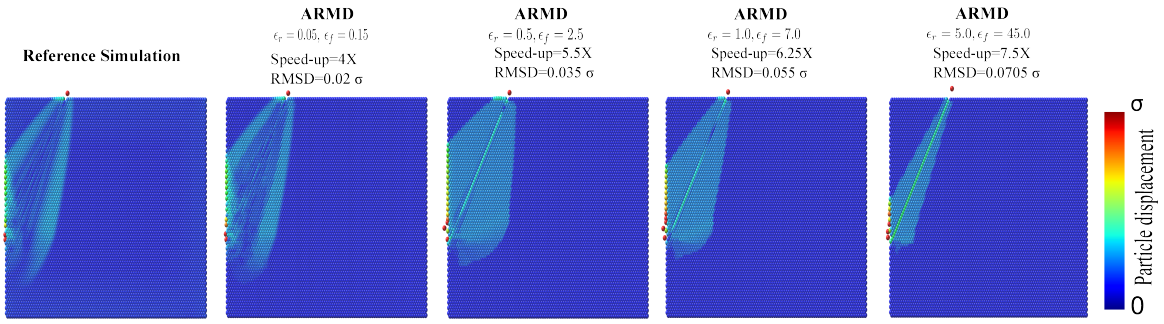


Fig. 2.8 Collision Cascade: ARMD simulations collision cascade of the same system with different AR parameters. Particles are colored according to the displacement from their initial positions. AR simulations in the NVE ensemble make it possible to finely trade between speed-up and precision.

applied an external force in the Y direction to model an electrochemical gradient across the membrane. Figure 2.9 represents the toy model. We divided the system into three types of particles:

1. Type 1: particles that represents an ion. The initial velocity of these particles is set in the Y direction only (red color particles in Figure 2.9).

2. Type 2: channel particles, *i.e.* particles in close proximity to the passing ion (green particles in Figure 2.9).

3. Type 3: membrane particles (violet particles in Figure 2.9).

In this system, containing 12,141 particles, Type 1 particles enter into the pore formed by Type 2 particles, and Type 2 particles either accelerate or decelerate Type 1 particles. Since we were interested in the motion of Type 1 particles (*e.g.* the speed at which they traverse the channel), we did not apply any restraint on these particles ($\varepsilon_r = \varepsilon_f = 0$). For Type 2 particles, we set ε_f to two, four, six, eight and ten times of ε_r values. For Type 3 particles, AR parameters were five times larger than Type 2 particles. Each simulation was performed for 150,000 steps, with a time step equal to 0.0001. Simulations were performed in the LJ units.

In order to verify the properties of the system, we also ran a reference MD simulation using with a Verlet integrator. For speed-up measurement, we ran each simulation 10 times and computed the average time spent in each integration step. We measured the speed-up with respect to the reference simulation. We computed the probability density of type 1 particle along the Y direction (the channel axis) [83]. Figure 2.10 shows the obtained speed-up with respect to the ε_r values of Type 2 particles. We achieved a 10X speed-up with $\varepsilon_f = 2 * \varepsilon_r$, and

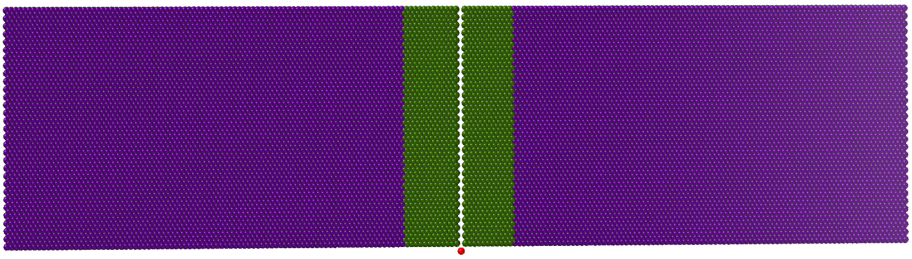


Fig. 2.9 Toy model of an ion passing through a channel (12,141 particles). This system contains 3 types of particles. The red particle is always active, green particles are less restrained compared to violet particles.

a 6-7X speed-up with other ratios of AR parameters. Higher speed-ups were observed with lower ratios of $\varepsilon_f/\varepsilon_r$, which we also observed in the collision cascade example. Figure 2.11 shows the probability density of an ion inside the toy system. From Figure 2.11, we can infer that, except for the largest ratio in AR parameters ($\varepsilon_f/\varepsilon_r = 10$), all other simulations retain the same distribution as the reference simulation, hence the same ion circulation dynamics, while still allowing for large speed-ups.

2.3.4 Toy model of a solvated polymer

In order to demonstrate how ARMD may be used to estimate the statistical properties of a system in the NVT ensemble, we performed a simulation of a polymer in a cubical solvent box. The toy polymer contains a chain of 8 identical particles with mass 10 grams/mole. The solvent box (length 50) contains 4,000 Lennard-Jones particles with mass 2.9 grams/mole. A harmonic potential was used for bonded interactions (bond, angle and dihedral terms), and a Lennard-Jones potential (cut-off 12.5) was used for non-bonded interactions. The system was initially minimized and then simulated in the NVT ensemble using a $1fs$ time step. Initial velocities were generated using the Maxwell-Boltzmann distribution at a given temperature. Periodic boundary conditions were employed in all three directions. Since we wanted to compute statistic averages of the polymer, we did not apply any restraint on it ($\varepsilon_r = \varepsilon_r = 0$), while solvent particles were restrained with $\varepsilon_r = 0.25Kcal/mol$ and $\varepsilon_f = 2.50Kcal/mol$.

We simulated this system for different temperatures (300K, 400K, 500K, 600K and 700K). Figure 2.12 shows that the system takes few time steps to reach the desired temperature. For each temperature and combination of AR parameters, we observed an equilibrium in the number of active particles. The AR parameters also influence the time needed to reach this equilibrium and the desired temperature (Figures 2.13, 2.14, 2.15 and 2.16).

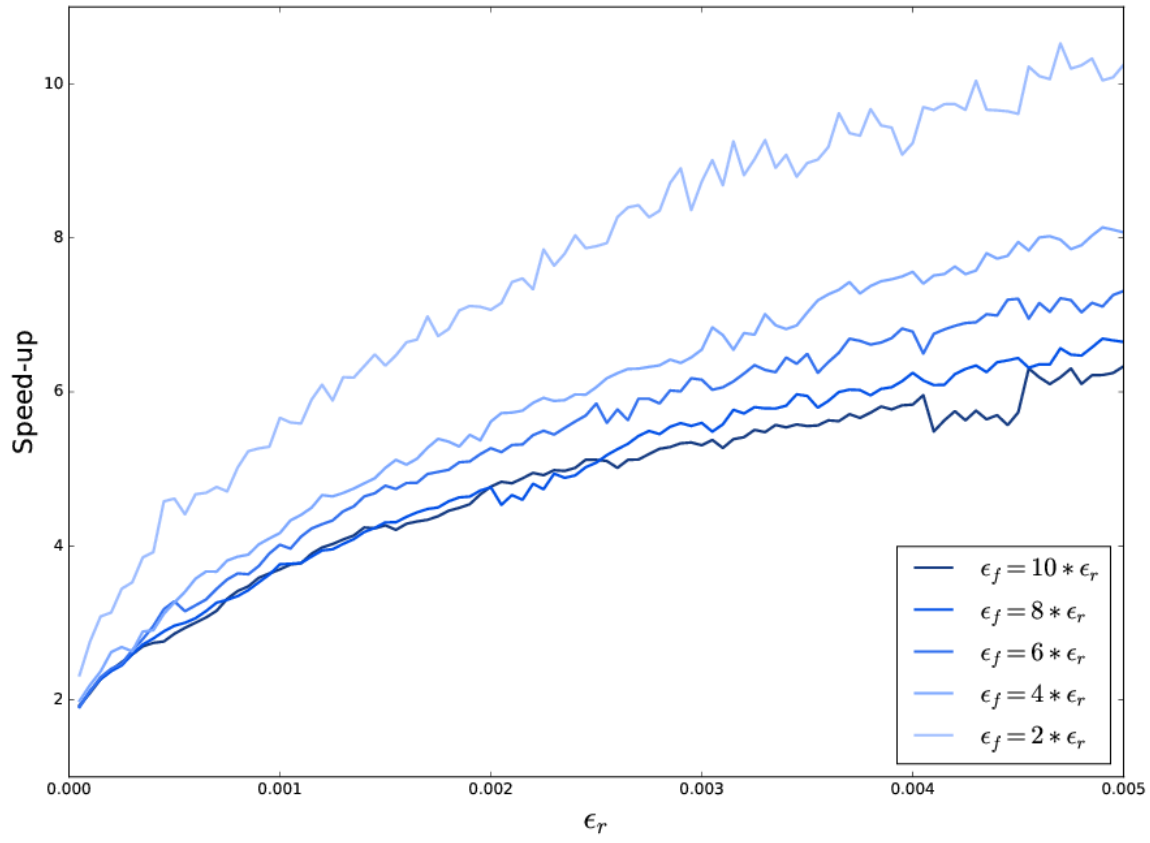


Fig. 2.10 Toy model of an ion passing through a channel (12,141 particles). Speed-up as a function of the restrained parameter ε_r for Type 2 particles.

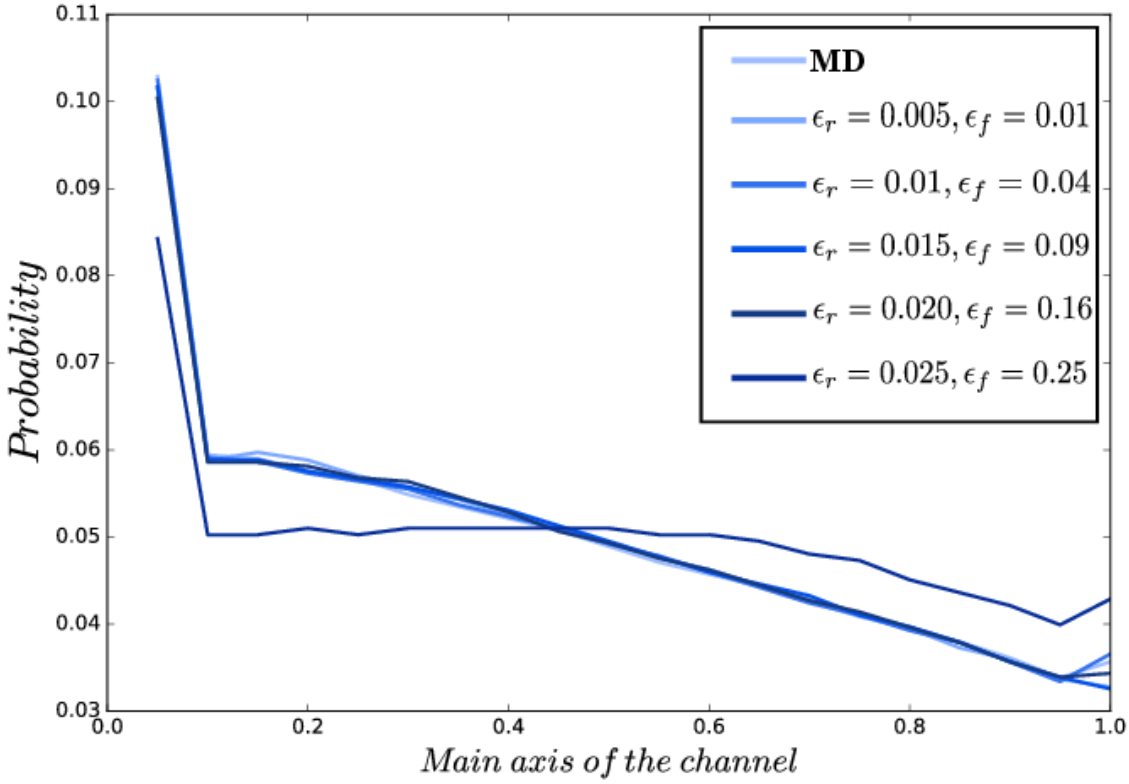


Fig. 2.11 Toy model of an ion passing through a channel (12,141 particles). Differences in the ion probability density along the channel axis from the reference MD. Different plots represent different ratios of AR parameters.

In order to verify statistical averages, we compared the radial distribution function (RDF) obtained by 10 *ns* of ARMD simulation with the one obtained by molecular dynamics, in the NVT ensemble at 300 K temperature, with the aforesaid parameters. On average, 36% of the particles were active during the ARMD simulation. Figure 2.17 shows that the RDF obtained by ARMD matches the one obtained with MD, indicating that statistical averages of position-dependent quantities are not modified [79].

In order to understand the effect of the number of active particles on temperature in ARMD, we performed three simulations with varying ε_r and ε_f values at 300K. Figure 2.18 shows that different ε_r and ε_f values lead to different numbers of active particles for the desired temperature. Figure 2.18 and table 2.1 illustrate how fluctuations in temperature are also related to the number of active particles, and fluctuations of the number of active particles: higher percentages of restrained particles lead to higher temperature fluctuations.

In the NVT ensemble, the overall achievable speed-up when computing a statistical average depends both on the instantaneous computational speed-up that can be achieved at each time step, and the modification of the variance caused by the AR Hamiltonian [84]. We refer the reader to Artemova and Redon [79] for an example analysis of the speed-up that can be achieved in the NVT ensemble.

$\varepsilon_r(Kcal/mole)$	$\varepsilon_f(Kcal/mole)$	$\% < N_{res} >$	avg. Temp. (K)	*St.dev.*
0.25	2.50	63.5	301.94	12.302217
1.3	4.5	90.0	299.501	22.913903
3.5	4.69	98.8	302.456	78.788141

Table 2.1 Table represents the average and standard deviation in temperature obtained by 3 different ARMD simulations with different AR parameters.

2.4 Conclusion

We have presented novel algorithms enabling the use of AR molecular dynamics simulations with the LAMMPS software package, and we have demonstrated how ARMD makes it possible to speed up simulations.

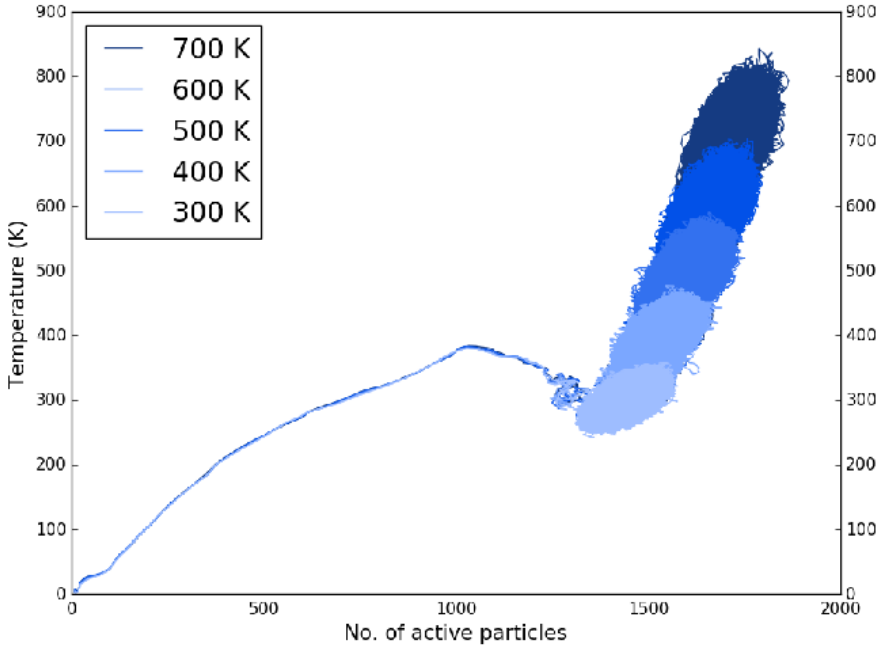


Fig. 2.12 Variation of temperature with respect to the number of active particles.

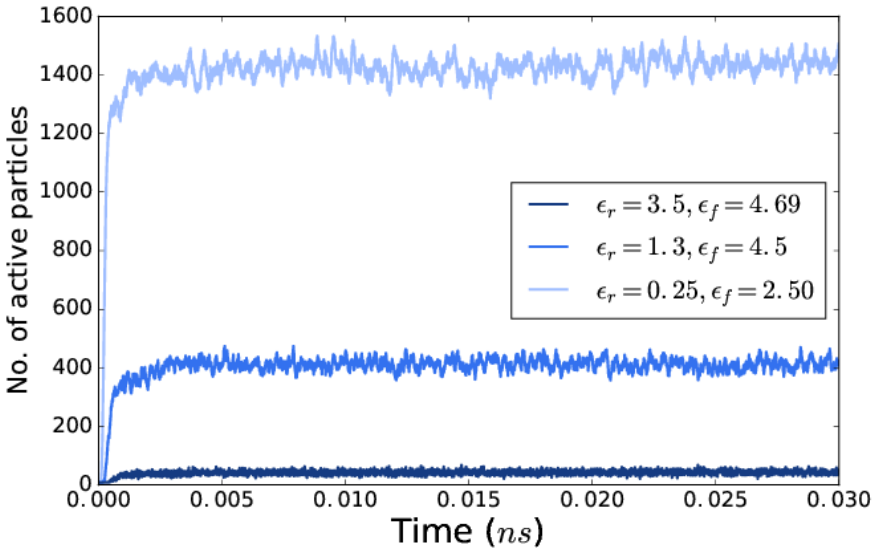


Fig. 2.13 Equilibrium period for the number of active particles.

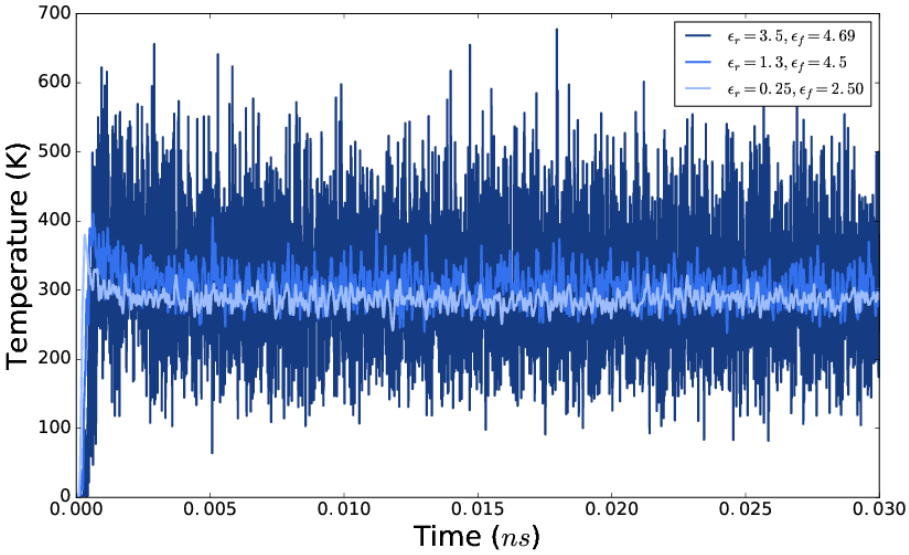


Fig. 2.14 Equilibrium period for the instantaneous temperature.

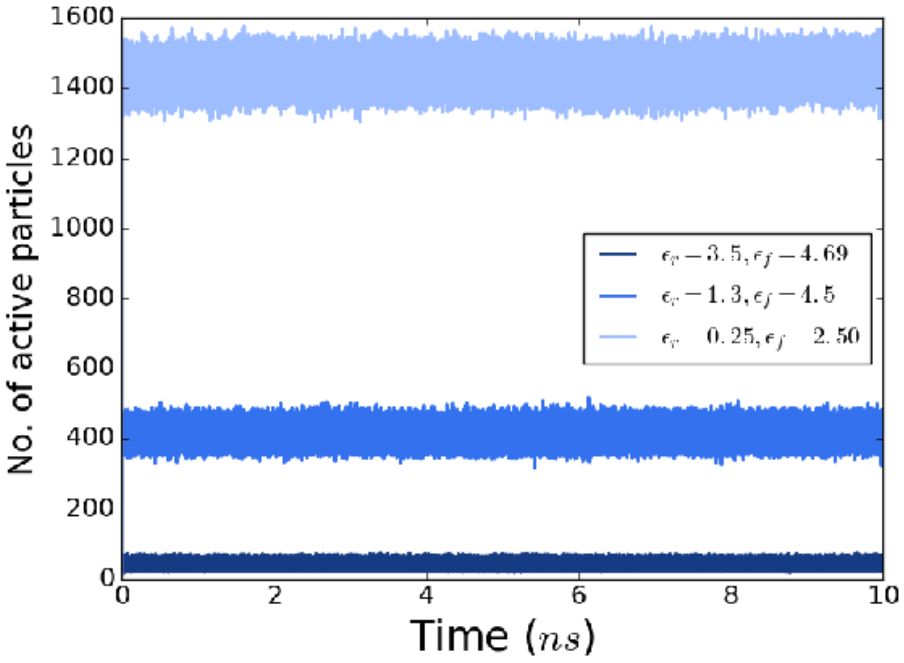


Fig. 2.15 Time evolution of the number of active particles.

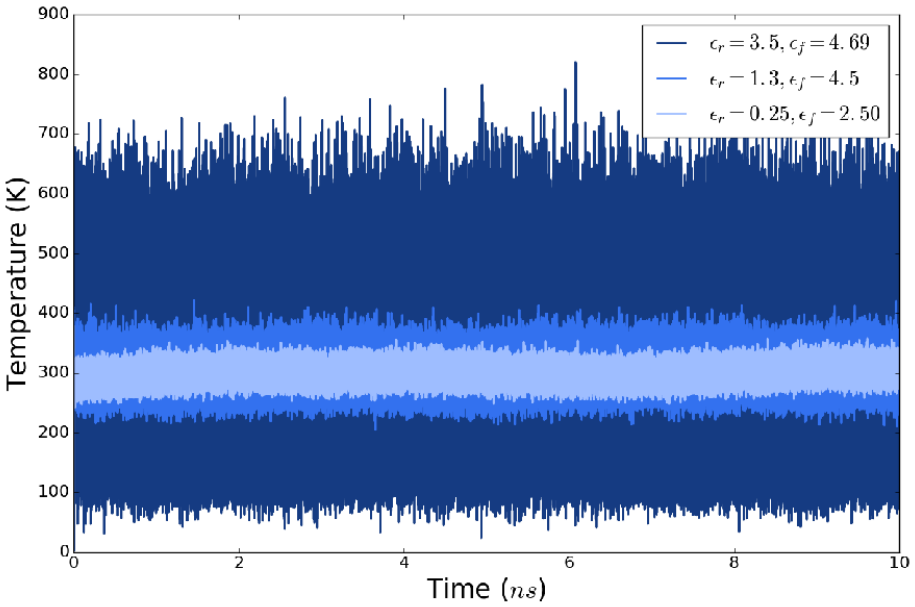


Fig. 2.16 Instantaneous temperature of the system with different AR parameters.

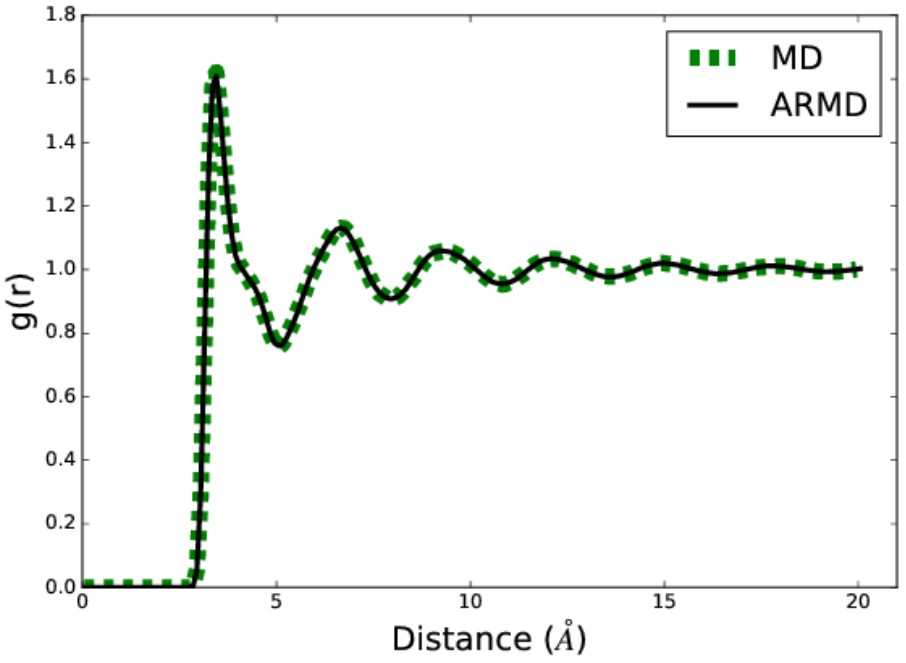


Fig. 2.17 Radial distribution functions of polymer obtained with ARMD and with MD

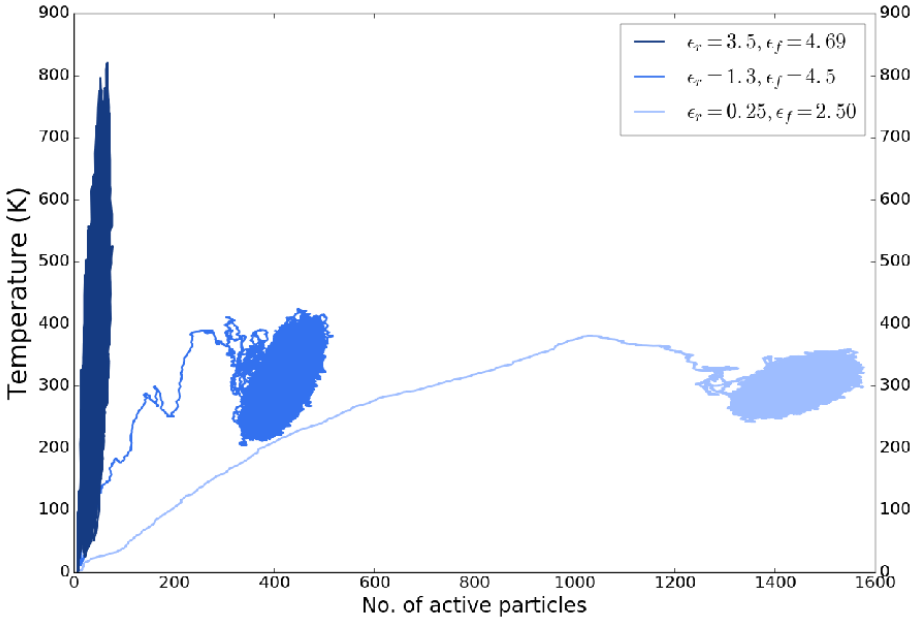


Fig. 2.18 Temperature profile of the system with respect to the number of active particles with variation in ε_r and ε_f values.

Chapter 3

Single-pass Incremental Force Algorithm for ARMD

Abstract

Adaptively Restrained Molecular Dynamics (ARMD) simulations reduces the number of force computations by adaptively switching on and off positional degrees of freedom of particles, and hence, more number of integration steps can be performed in wall clock time. In order to gain speed, ARMD uses active neighbor list (ANL) and incremental force update algorithm, that speed-up the simulations. However, these algorithms enforce the limitation of having at-lease 60% of the particles with positional degrees of freedom off. This chapter presents new algorithm *Single-pass incremental force update* algorithm that will overcome the aforementioned limitation. Moreover, the proposed algorithm is assessed and validated on four different benchmarks using different potentials in both NVE and NVT ensembles. Our results in NVE and NVT ensembles show that ARMD can be used to measure statistical properties faster, and in canonical ensemble, ARMD not only reproduces the statistical averages obtained by conventional MD but also speed up the convergence of an observable in Wall clock time.

3.1 Introduction

In previous chapter, we introduced the ANL and incremental force update algorithms. The incremental force update algorithm required two passes: one to subtract interactions involving from the previous time step and second to add updated interactions for the current time step. Each pass involves computation of force increments and hence, such incremental algorithms require force increments to be calculated twice in a single integration step, which acts as a bottleneck in ARMD simulations. In this chapter, we present a *single-pass incremental force update algorithm* for ARMD that will overcome the aforementioned limitation. Furthermore, we will use the ANL, incremental force update algorithms and the Wolf method for computing electrostatic interactions [85–90].

3.2 Algorithms for AR Molecular Dynamics

Let C denote the complete list of particles, A denote the list of active particles, and R denote the list of restrained particles. Let S_A (resp. S_R) denote the list of particles that *switched* to being active (resp. restrained) between the previous and current time step.

In ARMD, a system of N particles can be referred to as a combination of N_A active and N_R $(= N - N_A)$ restrained particles. Thus, at any time-step, interactions between particles can be categorized as *active interactions* (interactions involving at least one active particle) and restrained interactions (interactions between restrained particles only). Precisely, active interactions involve forces between either two active particles (F_{AA}) or one active particle and one restrained particle (F_{AR}), while restrained interactions involve forces between two restrained particles (F_{RR}). Figure 3.1 shows the three types of interactions on an example.

Most often, forces acting on particles only depend upon their relative positions. At a given time step, relative positions between restrained particles do not change; hence, restrained interactions and associated forces remain unchanged. This eliminates the need to compute F_{RR} force components involving restrained interactions, and the F_{RR} force components from the previous time step can be cached and reused. However, active particles update their positions, and the associated force components (F_{AA} and F_{AR}) need to be updated at each time step.

3.2.1 Active Neighbor List

In order to compute forces due to active interactions, we previously introduced Active Neighbor Lists (ANLs) *i.e.* lists of active neighbor particles at a given time-step [81]. The ANLs are constructed from Full Neighbor Lists (FNLs), and the ANL of an active particle

i can be described as $ANL(i) = \{\forall j \in FNL(i) : j \in R \cup (j \in A \cap (j < i))\}$. In other words, $ANL(i)$ contains all the restrained neighbors of i, and the active neighbors of i with smaller indices (due to the condition $j < i$).

3.2.2 Two-pass incremental force update algorithm

In MD, the forces acting on all particles are typically computed once, after updating particles positions. In contrast, in ARMD, only the positions of active particles are updated, and forces are updated based on the new positions of active particles, thereby eliminating the need to compute all forces. In our previous approach, active interactions were incrementally updated using an algorithm that involved two force update steps[82, 79, 81]:

1. **First pass** or subtraction step: This pass removes the force increments involving active particles based on the old positions, by computing the force increments due to active particles and subtracting them from the total force.

2. Position update step: This involves updating the positions of active particles only, instead of updating the positions of all particles.

3. **Second pass** or addition step: This step involves computing force increments based on the updated positions of active particles, which are then added to the forces obtained from the subtraction step.

This incremental force update algorithm requires force increments to be calculated twice in a single integration step, limiting the usefulness of ARMD to cases where at least 50% of particles are restrained [81]. Furthermore, if all particles are active, the proposed incremental force algorithm would be twice times slower than the normal MD algorithm.

3.2.3 Single-pass incremental force updates

In order to overcome the aforementioned limitation, we have designed a *single-pass* incremental force computation algorithm.

In ARMD, the force F_i acting on a particle i can be written as $F_i = F_{i_{AA}} + F_{i_{AR}} + F_{i_{RR}}$ (figure 3.1), where $F_{i_{AA}}$ refers to forces between two active particles, $F_{i_{AR}}$ refers to forces between an active and a restrained particle and $F_{i_{RR}}$ refers to the force between two restrained particles.

Assume there are no switching particles between the n-th and $(n+1)$-th time step. Using the two-pass incremental force computation algorithm, the position of an active particle i is updated, and the total force $F_i = F_{i_{AA}} + F_{i_{AR}}$ applied to it is updated during the addition step.

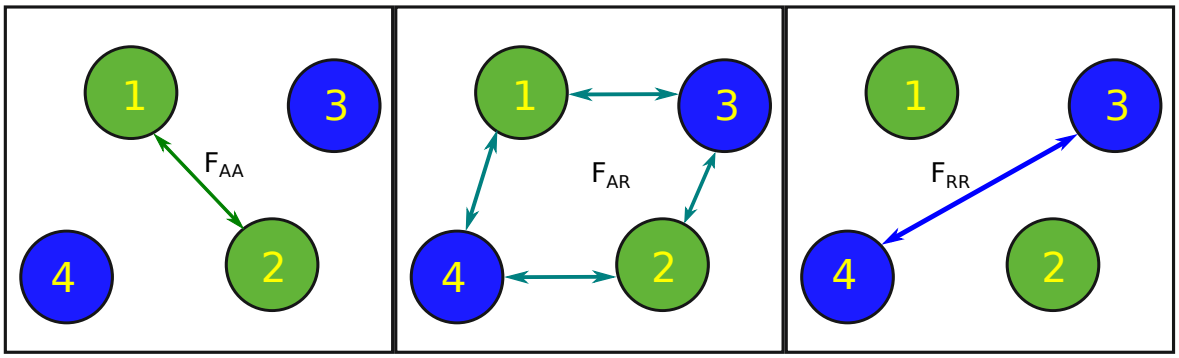


Fig. 3.1 Different types of force components present in the system. Green particles are active and blue particles are restrained. The force components F_{AA} , F_{AR} and F_{RR} represent forces acting between active particles, forces between active and restrained particles, and forces between restrained particles.

Since the new position is not updated *after* the addition step, though, the next substraction step substracts the *same* force increments, yielding $F_i = 0$ after the subtraction step. Therefore, in case of an active particle, the subtraction step can be replaced by assigning zero to the associated force.

In case of a restrained particle i, the total force applied to it is $F_i = F_{i_{AR}} + F_{i_{RR}}$. As noted above, $F_{i_{RR}}$ can be cached and reused, so that we only have to compute $F_{i_{AR}}$. In single-pass algorithm, force on the particle i is assigned as $F_i = F_{i_{RR}}$ before computing the force component, unlike the previous algorithm that adds and removes $F_{i_{AR}}$ force component (as the $F_{i_{RR}}$ force component remains unchanged and could be retained from the previous time step). Also, this algorithm eliminates the need to carry out subtraction step and subtraction step can be removed with algorithm 8. However, the time evolution of the system switches the state of the particles, force components associated with these switched particles need to be computed, before applying algorithm 8. Particles can switch their states in two ways:

1. Switching to active state
2. Switching to restrained state

It is likely that the switching in the state of particles may cause the change in the type of interaction (to an active or restrained interaction) and hence the associated force components.

An active interaction associated with pair $i-j$ can switch to a restrained interaction in the following two ways:

- Assuming at time step n both particles i and j are active and they switch to a restrained state at $n+1$ time step, the F_{AA} force component associated with the pair $i-j$ would now switch to the F_{RR} force component. As ANLs avoid computation of F_{RR} force

component, this switched force component needs to be computed and stored for next time steps.

- On the other hand, if we assume particles i and j to be active and restrained respectively at a time step n (associated force component has type F_{AR}), and at $n+1$ time step the particle i changes to a restrained state, this switching updates the interaction type from active to restrained and associated force component from F_{AR} to F_{RR}. As mentioned previously, the updated F_{RR} force component needs to be computed and stored for the next time-steps.

Similarly, a restrained interaction belonging to a pair $i-j$ can switch to an active interaction in the following ways:

- In case, particle i or j switches their state from restrained to an active state, force component F_{RR} associated with this pair switches to F_{AR}. This updated F_{AR} force component needs to be computed and subtracted from the forces of restrained neighbors of the this switched particle i.

- Another possibility wherein both particles i and j switch from restrained to an active state, the associated F_{RR} force component switches to the F_{AA} force components. This type of switching does not require any force computation as zero can be assigned for the forces associated with i and j.

In order to compute the switched force components (F_{RR} or F_{AR}), we used a modified active neighbor list ANL'. The ANL' of a switched particle i is an extracted ANL that contains only restrained neighbors. Algorithm 9 gives a description of extracting the ANL' from the ANL. Algorithm 10 shows the pseudo-code to compute switched force components. Line 3 of algorithm 10 computes F_{RR} force components and stores them in F^+ and line 9 computes F_{AR} force components and stores then in F^- for the switched particles.

Algorithm 8 gives the pseudo-code to perform the ARMD integration step using the ANLs and incremental force update algorithm. In this algorithm, before computing forces based on the new positions, zeros (line 3) and the force components F_{RR} (line 6) are assigned to the forces corresponding to active and restrained particles respectively.

3.3 Analysis

In this section, we assess different algorithms for constructing ANLs and incrementally update forces. We compare these algorithms with brute-force MD algorithms which have

Algorithm 8: *SinglePassIncrementalForceComputation*()

for $i \in C$ **do**
  if $i \in A$ **then**
    $F_i^+ \leftarrow 0$
  end
  else
    $F_i^+ \leftarrow F_i^+ - F_i^-$
  end
  $F_i^- \leftarrow 0$
end
$F \leftarrow F^+ + ComputeForces(A, ANL)$

Algorithm 9: $ExtractANL'(i)$

$ANL'(i) \leftarrow \emptyset$
for $j \in ANL(i)$ **do**
  if $(i \in R \text{ and } j \in R)$ $or(i \in A \text{ and } (j \in R \text{ and } j \notin S))$ **then**
    $ANL'(i) \leftarrow ANL'(i) \cup j$
  end
end

Algorithm 10: *SwitchedForces*()

for $i \in S_R$ **do**
  $ExtractANL'(i)$
  $F^+ \leftarrow F^+ + ComputeForces(i, ANL'(i))$
  $ClearANL(i)$
end
for $i \in S_A$ **do**
  $BuildANL(i)$
  $ExtractANL'(i)$
  $F^- \leftarrow F^- + ComputeForces(i, ANL'(i))$
end

Algorithm 11: ARMD integration step

```
1  Update momenta
2  Update A, R, S_A, S_R
3  if (UpdateNeeded) then
4      for i ∈ C do
5          if i ∈ A then
6              Build FNL(i) and ANL(i) simultaneously
7          end
8          else
9              BuildFNL(i)
10         end
11     end
12 end
13 else
14     SwitchedForces()
15 end
16 Update Positions
17 SinglePassIncrementalForceComputation()
```

a running time equal to $\tau_{MD} = N * \tau_{FNL} + N * \tau_F$, where N is the total number of particles, τ_{FNL} is the time needed to build the FNL of one particle, and τ_F is the time required to compute the total force acting on one particle due to its neighbors. To simplify the analysis, we consider the timing required to construct the FNL rather than the HNL.

3.3.1 Time complexity

In this algorithm, the FNLs of all particles and the ANLs of active particles were constructed at the same time. The ANLs of particles that become active were constructed using the FNL of the corresponding particle, as building the ANLs from scratch is relatively more time consuming. The force components F_{AR} or F_{RR} for switched particles were then computed in two steps (Algorithm 10). The first step involves extracting ANL' from ANL, and the second step involves computing specific force components with ANL'.

The computation time for algorithm 11 is given by:

$$\tau_{ARMD1} = N * \tau_{FNL} + N_A * \tau_F + N_{SA} * \tau_{FANL} + N_{SA} * \tau_{SF} + N_{SR} * (\tau_{SF} + \tau_{CL}) + N_S * \tau_{Ex} \quad (3.1)$$

τ_{Ex} : Time to build the ANL' for one particle.
τ_{FANL} : Time to construct the ANL from the FNL.

τ_{SF} : Time to compute specific force components (F_{AR} or F_{RR}).
τ_{CL} : Time to clear an ANL.
N_{SA} : Number of particles switching to an active state.
N_{SR} : Number of particles switching to a restrained state.
N_S : Total number of switched particles ($N_{SA} + N_{SR}$).
Note that Equation (3.1) does not include τ_{ANL}, the time to compute an ANL, since, when neighbor lists are update from scratch (lines 3-12), Algorithm 11 computes FNLs and ANLs simultaneously.

Algorithm 11 is more efficient than classical MD when $\tau_{ARMD1} < \tau_{MD}$, *i.e.*:

$$N_A * \tau_F + N_{SA} * \tau_{FANL} + N_{SA} * \tau_{SF} + N_{SR} * (\tau_{SF} + \tau_{CL}) + N_S * \tau_{Ex} < N * \tau_F. \quad (3.2)$$

To simplify the analysis, we assume the following worst cases:

Assumption 1: All switching particles switch to an active state, making it necessary to construct *ANLs* for all switched particles: $N_{SR} = 0$ and $N_S = N_{SA}$.

Assumption 2: Computing a force component F_{AR} or F_{RR} for a particle takes as much time as computing the total force on this particle: $\tau_{SF} = \tau_F$.

Assumption 3: Even though a) computing the *ANL* of a particle from its *FNL* and b) extracting the ANL' of a particle from its *ANL* are both much more efficient than computing the *FNL* of a particle, we assume that the sum of τ_{FANL} and τ_{Ex} is equal to τ_{FNL}: $\tau_{FNL} = \tau_{FANL} + \tau_{Ex}$.

Considering these three assumptions, inequality 3.2 holds when:

$$N_A * \tau_F + N_S * \tau_{FNL} + N_S * \tau_F < N * \tau_F, \quad (3.3)$$

i.e. when:

$$\frac{N_A}{N} + \frac{N_S}{N}(1 + \frac{\tau_{FNL}}{\tau_F}) < 1. \quad (3.4)$$

3.3.2 Optimization

Although Algorithm 11 computes the *ANLs* and $ANL's$ required to perform the single-pass force update algorithm, it computes the *FNLs* of all particles when neighbor lists are updated from scratch. Since the *FNL* of a restrained particle is only useful when it switches to an active state, however, Algorithm 11 may be optimized by maintaining *hasFNL*, a list of particles for which the *FNL* has been computed, and using this list to determine when to update the various neighbor lists.

Algorithm 12 describes the optimized version. When a particle switches to a restrained state, the ANL of this particle is cleared, while retaining its FNL. When a particle switches to an active state and it does not have a FNL, its ANL and FNL are constructed *on-the-fly* (*i.e.* outside the neighbor list construction step). When a particle switches to an active state and already has an FNL (such as when a particle switches states multiple times between two neighbor list construction steps), the ANL of this particle is constructed from the existing FNL. Algorithm 13 shows how Algorithm 10 is modified in the optimized case, when *hasFNL* is available.

Algorithm 12: ARMD integration step

```
1  Update momenta
2  Update A, R, S_A, S_R
3  if (UpdateNeeded) then
4      hasFNL ← ∅
5      for i ∈ A do
6          Build FNL(i) and ANL(i) simultaneously
7          hasFNL ← hasFNL ∪ {i}
8      end
9  end
10 else
11     SwitchedForces'()
12 end
13 Update Positions
14 SinglePassIncrementalForceComputation()
```

The computation time for Algorithm 12 is given by:

$$\tau_{ARMD2} = N_A * \tau_{FNL} + N_A * \tau_F + N'_{SA} * \tau_{FNL} + (N_{SA} - N'_{SA}) * \tau_{FANL} + N_{SA} * \tau_F + N_{SR} * (\tau_{SF} + \tau_{CL}) + N_S * \tau_{Ex}, \tag{3.5}$$

where N'_{SA} is the number of particles without a FNL that have switched to an active state.

In order to compare the time complexity of Algorithm 12 with the MD algorithm, we considered the same assumptions as in the Algorithm 11) ($N_{SR} = 0$, $\tau_{FANL} + \tau_{Ex} = \tau_{FNL}$ and $\tau_F = \tau_{SF}$), and added another worst-case scenario assumption, where all switching particles switch to an active state and do not have an FNL: $N_S = N_{SA} = N'_{SA}$.
After substitution, Equation 3.5 changes to:

$$\tau_{ARMD2} = N_A * \tau_{FNL} + N_A * \tau_F + N_S * \tau_{FNL} + N_S * \tau_F + N_S * \tau_{Ex}. \tag{3.6}$$

Algorithm 13: $SwitchedForces'()$

```
for i ∈ S_R do
    ExtractANL'(i))
    F^+ ← F^+ + ComputeForces(i, ANL'(i))
    ClearANL(i)
end
for i ∈ S_A do
    if i ∈ hasFNL then
        BuildANL(i)
    end
    else
        Build FNL(i) and ANL(i) simultaneously
        hasFNL ← hasFNL ∪ {i}
    end
    ExtractANL'(i)
    F^- ← F^- + ComputeForces(i, ANL'(i))
end
```

Since both N_S and τ_{Ex} are small, we may neglect their product. As a result, Algorithm 12 is more efficient than classical MD when $N_A + N_S \leq N$, which is generally true for ARMD. The difference in the computation times of Algorithms 11 and 12 is:

$$\tau_{ARMD1} - \tau_{ARMD2} = (N - N_A - N_{SA}) * \tau_{FNL}. \tag{3.7}$$

If the sum of the number of active particles and the number of particles that switch to an active state is smaller than the total number of particles (which is true most of the time since the number of switching particles is small), then Algorithm 12 performs better than Algorithm 11.

3.4 Results and Discussions

The suggested algorithms were validated on the following benchmarks:

1. Systems of Lennard-Jones particles.
2. Simulation of Crystal NaCl.
3. High-velocity impact of nanodroplet.
4. Simulation of a polymer in the solvent.

3.4.1 Systems of Lennard-Jones particles

We performed simulations of different numbers (500, 4000 and 108000) of Lennard-Jones particles using the AR integrator in the NVE ensemble. All simulations were carried out in reduced units (lj units) using the LAMMPS MD package. Particles were generated on a fcc lattice with density 0.8442 and initial velocities were assigned according to the Boltzmann distribution at temperature $k_BT = 1.44$. For all simulations, we used a time-step of 0.005 and interactions beyond a distance of 2.5σ were ignored. Periodic boundary conditions were employed in all three directions.

We performed two series of benchmarks. In the first series, the percentage of restrained particles was constrained by assigning very high AR parameters to a specific number of particles (and zero AR values to the others), to ensure they would not switch their state and remain restrained. In the second series, we let particles switch their state and assigned all particles the same (lower) AR parameters.

We compared the run time of our algorithms with that of LAMMPS algorithms. The reference simulations were performed using LAMMPS neighbor lists and force algorithms, whereas ARMD simulations were performed using the ANLs and single-pass incremental force algorithms. The neighbor list was updated at every 20 time-steps. For speedup measurements, we computed the average time spent at each integration step and the time spent in the construction of the ANL as well. These values were then compared to the average time spent per integration step and per construction of neighbor list in the reference simulations.

Series 1 (constrained number of restrained particles): Figure 3.2 shows the achieved speedup in constructing the ANL vs. the percentage of restrained particles. The construction of both the ANL and the FNL required a $27r_c^3$ volume to be searched, in contrast to the HNL, which required half of that volume. As a result, the construction time for building the ANL or the FNL was twice that of the HNL. However, in order to have a speedup when constructing the ANL, at least 50% of the particles are required to be restrained. Also, we found that the obtained speedup in constructing the ANL was the same regardless of the number of particles present in the systems. In conclusion, we observed a 2X (resp. 4X) speedup in constructing the ANL while restraining 80% (resp. 90%) of the particles.

Figure 3.3 shows the achieved speedup per integration step with respect to the percentage of restrained particles in the system. Thanks to the new single-pass force update algorithm, there is no constraint on the minimum number of restrained particles. In fact, even with only 40% of particles restrained, we achieved a $1.5X$ speedup. With 60% of the particles

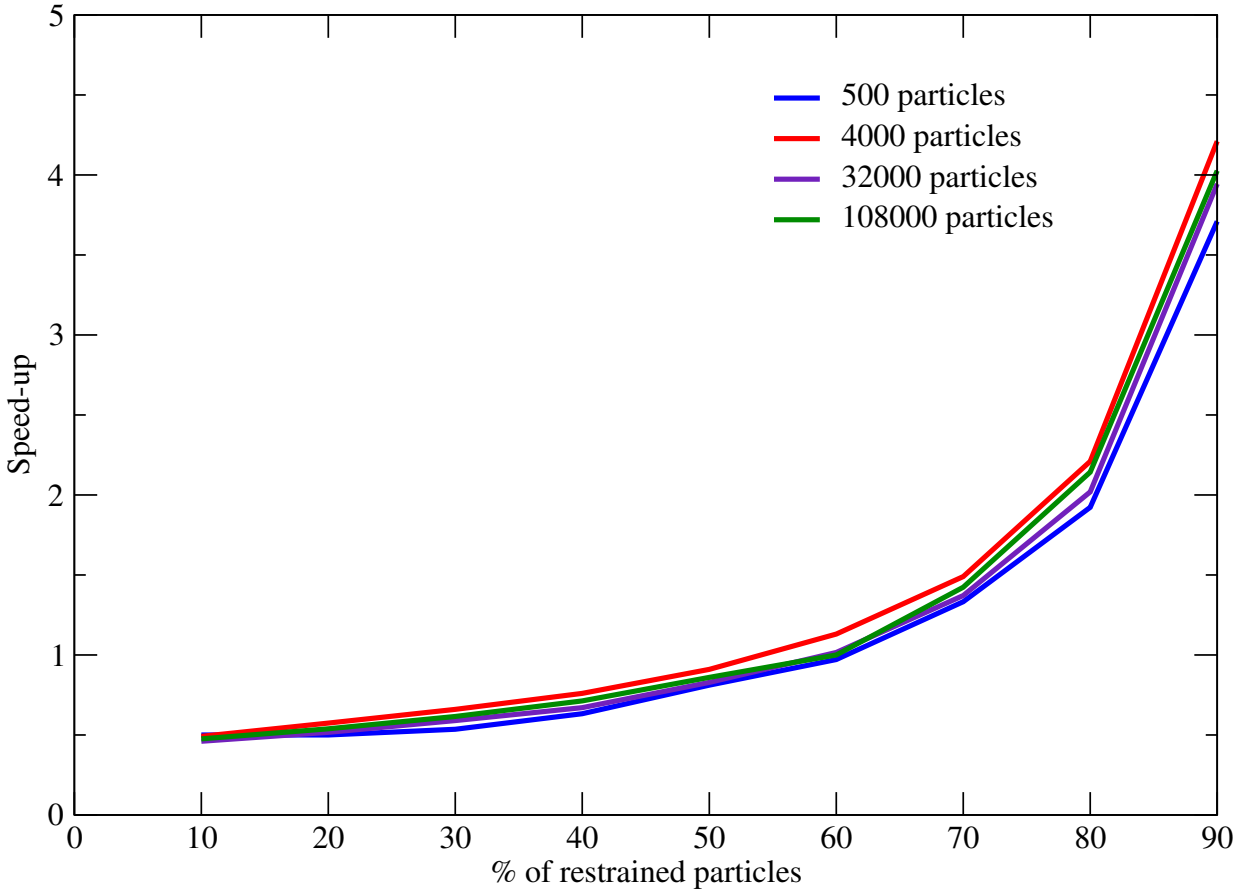


Fig. 3.2 Obtained speed-up in construction the ANL (compare to the HNL) as a function of percentage of restrained particles.

restrained, our algorithm performed twice as fast as the LAMMPS algorithm. Furthermore, 3X to 5X speedups were observed when $80-90\%$ particles were restrained.

Finally, we measured the overall speedup which encompasses the time to built the NLs and the time to perform integration steps. Figure 3.4 shows the overall speedup vs. the percentage of restrained particles. For less than 50% of restrained particles, the new single-pass incremental algorithm is twice faster than the two-pass incremental algorithm. As expected, the single-pass incremental algorithm is always faster than the two-pass incremental algorithm due to the reduced number of force computations. When comparing with traditional MD and the number of restrained particles was less than 20%, however, no speedup was achieved. This is explained by the cost of constructing the ANL, which is approximately twice the cost of constructing the HNL. Overall, A similar trend to the speedup in Figure 3.3 was observed with the overall speedup: a 2X speedup was observed with 60% particles restrained and up to 5X speedup with 90% particles restrained (Figure fig:case1overallspeedup).

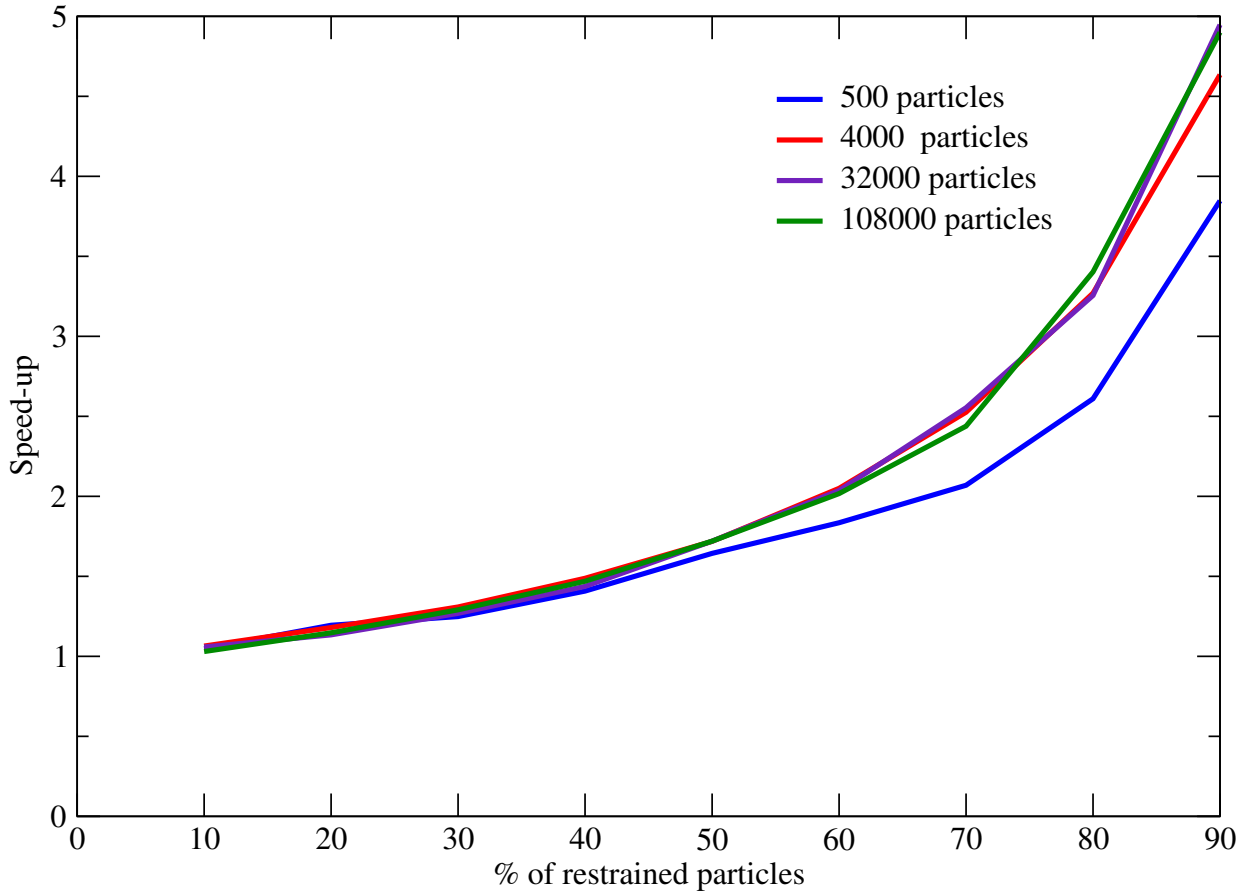


Fig. 3.3 case 1: Obtained speed up in performing integrations step.

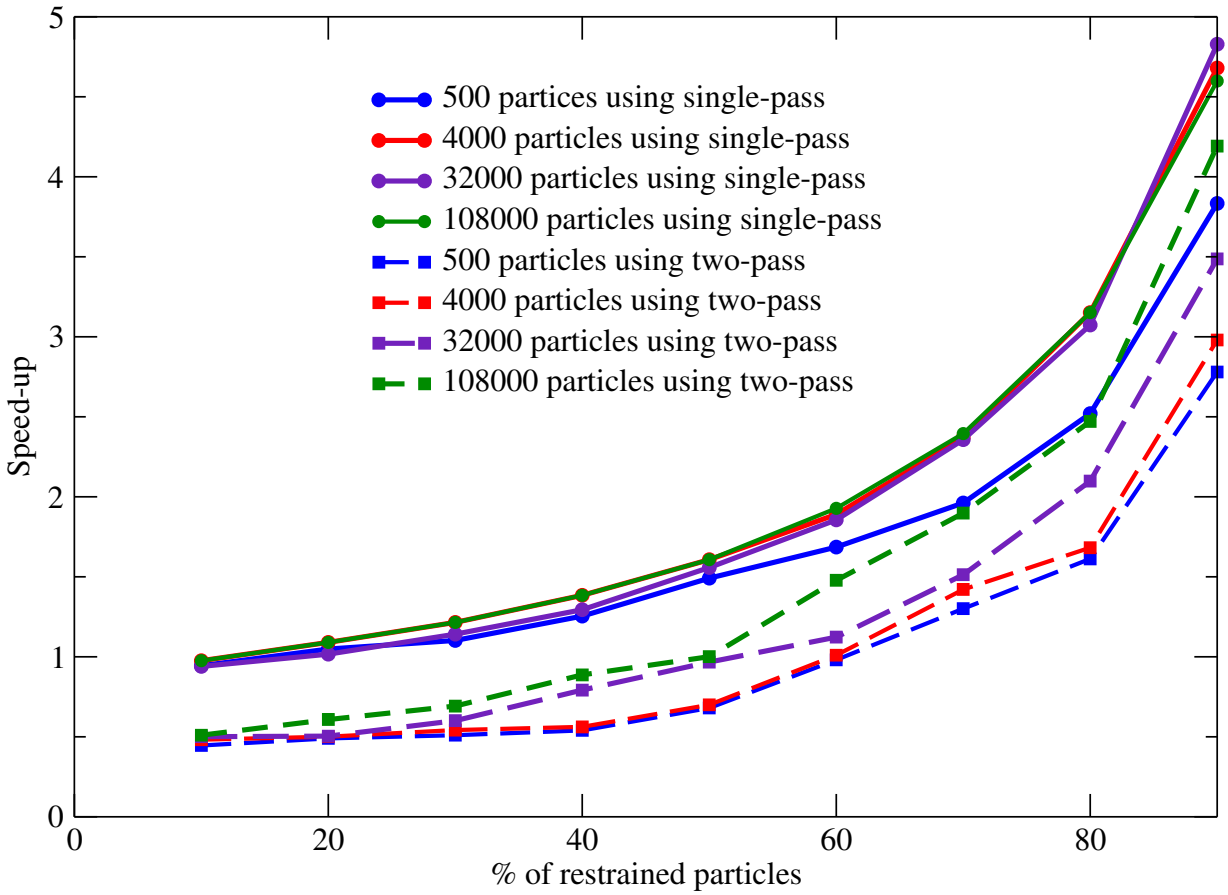


Fig. 3.4 Series 1: Overall speedup obtained as a function of the percentage of restrained particles using the single-pass and two-pass incremental algorithms. The new single-pass algorithm always performs better than the two-pass algorithm.

ε_r	ε_f	500		4000		32000		108000	
		$\% < N_R >$	$\% < N_S >$	$\% < N_R >$	$\% < N_S >$	$\% < N_R >$	$\% < N_S >$	$\% < N_R >$	$\% < N_S >$
0.5	1	8.21	0.015	36.34	0.0075	28.12	0.0011	26.97	0.0054
1	2	26.47	0.024	56.94	0.0095	36.53	0.0091	45.73	0.0041
1.5	2	57.48	0.034	82.015	0.0042	46.96	0.0084	67.99	0.0073
2	4	60.54	0.029	84.27	0.0024	57.24	0.0021	70.99	0.0011
2.5	4	68.04	0.025	87.23	0.0092	71.32	0.0062	77.81	0.0063
3	5	81.25	0.021	91.08	0.0055	79.14	0.0055	87.16	0.0077
3.5	5	86.81	0.018	96.23	0.0032	85.75	0.0031	93.36	0.0021
4	5	92.42	0.014	96.23	0.0085	98.34	0.0022	97.15	0.0015

Table 3.1 AR parameters for Lennard-Jones systems. The average percentage of particles that switch states at each time step is very small, and does not significantly affect performance.

Series 2 (identical AR parameters for all particles): In this series of benchmarks, particles were allowed to switch states during simulation. Table 3.1 shows that the average number of switched particles at each time step was much smaller than the number of particles in the system. Thanks to this, switched particles did not have much influence on the obtained speedup, which reproduced the patterns observed in Series 1, and the single-pass force update algorithm always performed better than the two-pass incremental algorithm.

For the system containing 500 particles, with the single-pass incremental algorithm, a 2.1X to 3.5X speedup was observed with 81% to 86% particles restrained, and a maximum speedup of 4.5X was achieved when 92% of the particles were restrained. The AR parameters, the average number of switched particles and the average number of restrained particles can be found in Table 3.1.

For the system containing 4000 particles, a 3.8X speedup was attained when 91% of the particles were restrained, whereas in Series 1 the corresponding speedup was 4.5X with the same number of restrained particles. When 96% and 98.5% of the particles were restrained, we achieved a 6X and 8.9X speedup respectively. A 1.2X speedup was observed with the 57% of the particles as restrained particles. In another system containing 32000 particles, a 2X speedup was achieved with 46% of restrained particles. A 4X to 6X speedup was observed with 85% to 98% of particles restrained. For the system containing 108000 particles, a maximum speedup of 7.5X was observed with 97% of restrained particles. The reduction in overall speedups in Series 2 as compared to Series 1 is due to switched particles (building ANL and computing force components), and the fact that the observed percentages of restrained particles are averages.

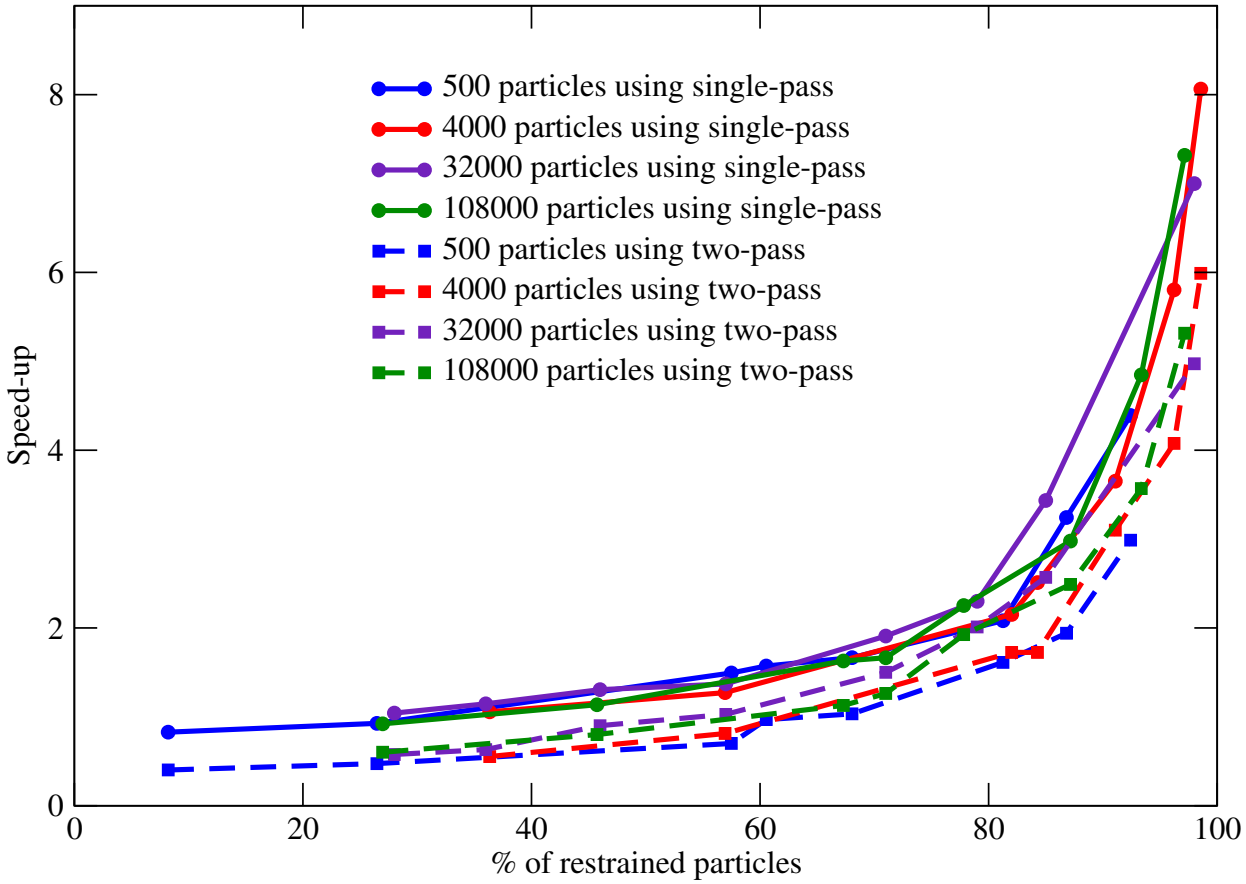


Fig. 3.5 Series 2: Speedup achieved with the single-pass and two-pass incremental algorithms over MD as a function of the percentage of restrained particles.

3.4.2 Simulation of Nacl

In order to validate our algorithm on a computationally expensive potential, we simulated a system containing 8000 (4000 Na^+ and 4000 Cl^-) NaCl. In this NaCl system, charged particles Na^+ and Cl^- interact via electrostatic interactions. In MD, during force calculations, most of the computational time is used in calculating electrostatic forces. In the NaCl system, around 90% of the total time is spent in the computation of electrostatic interactions.

The NaCl system was simulated using the Tosi-Fumi (TF) potential augmented with a coulombic potential. The TF potential is given by equation 3.8, and this potential is broadly used in simulation of alkali halides.

$$U(r) = A\ exp\left(\frac{\sigma - r}{\rho}\right) - \frac{C}{r^6} + \frac{D}{r^8} \qquad r < R_c \tag{3.8}$$

The first term in equation 3.8 is the Born-Mayer exponential repulsive term and the second term involves 8, 6 Van der Waals attractive interaction. Parameters for TF potential are taken from reference [91]. Instead of calculating electrostatic forces based on Ewald summation method, we used Wolf summation. The Wolf method is can approximation to compute electrostatic interactions efficient $(O(N))$ as compared to the Ewald-based methods. Furthermore, the Wolf method can use neighbor lists for computing electrostatic forces:

$$E_{Wolf} = \frac{1}{2}\sum_{i=1}^{N}\sum_{\substack{j\neq i \\ r_{ij}<R_c}}\left[\left(\frac{q_i q_j \operatorname{erfc}(\alpha r_{ij})}{r_{ij}} - \frac{q_i q_j \operatorname{erfc}(\alpha R_c)}{R_c}\right)\right] - \left(\frac{\operatorname{erfc}(\alpha R_c)}{2R_c} + \frac{\alpha}{\sqrt{\pi}}\right)\sum_{i=1}^{N}\frac{q_i^2}{4\pi\varepsilon_0} \tag{3.9}$$

In equation 3.9, erfc is the complementary error function, q_i and q_j represent the point charges on particles i and j, α is the damping parameter and R_c is the cut-off radius. Details regarding the Wolf method can be found from references [85–90]. For the Wolf method, we used the same parameters as those mentioned in the literature [92].

Cut-off distances of 7.5Å and 15Å were used for the TF potential and Wolf summation respectively. Initial velocities were assigned at 270K temperature. The system was simulated for 100000 time-steps using an integration time-step of $2fs$ in the NVE ensemble. In order to measure the speedup, we performed reference MD simulations as well as ARMD simulations with different AR parameters. For timing measurements, we ran 50 independent simulations (MD and different ARMD simulations) for each combination of parameters and timings were averaged over these 50 simulations.

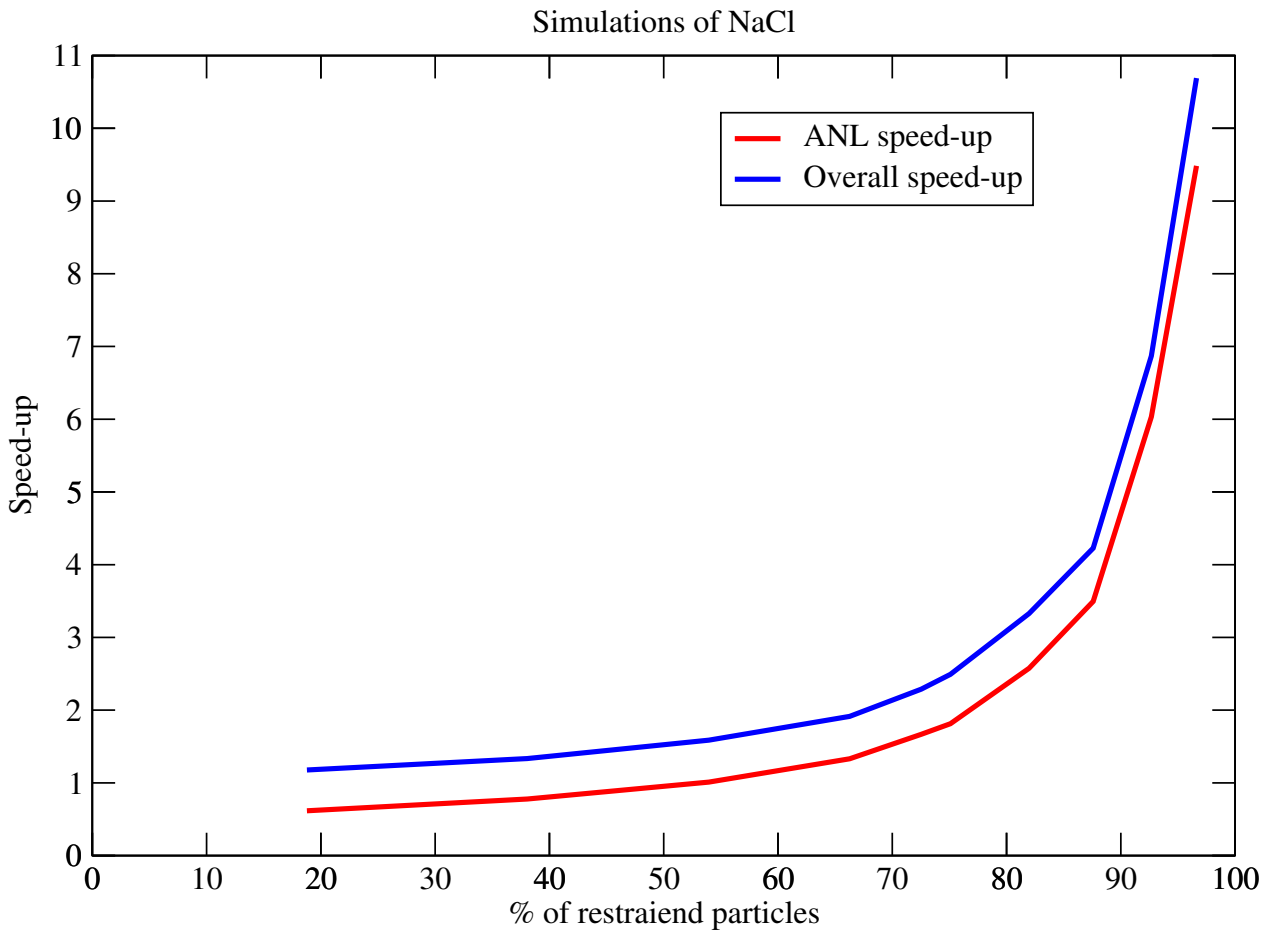


Fig. 3.6 Speedup achieved with ARMD for ANL construction (as compare to the HNL) and per time step in the NaCl benchmark.

ε_r	ε_f	$\% < N_R >$	$\% < N_S >$
0.1	1	18.78	0.0032
0.5	1	38.056	0.010
1	2	53.95175	0.0087
1.5	2	66.272125	0.0063
2	5	72.53125	0.0025
2.5	5	75.0875	0.0082
3	5	81.966625	0.0042
3.5	5	87.5818625	0.0072
4	5	92.67155	0.0065
4.5	5	96.6208125	0.0054

Table 3.2 shows the AR parameters used to simulate *NaCl*.

Different AR parameters (Table 3.2) give rise to different average numbers of active and switched particles. Figure 3.6 shows the speedup obtained with ARMD compared to MD. We achieved a 2X speedup with 65% of restrained particles, and a 4X to 10X speedup was achieved with 82% to 96% of restrained particles. This benchmark shows that combining the Wolf method with ARMD significantly reduces the number of calculations, which may result in a significant speedup.

3.4.3 High-velocity impact of nanodroplet

The high-velocity impact of a nanodroplet on a crystal surface changes the state of the crystal to an amorphous phase, and this process is known as amorphization [93, 94]. We performed this benchmark to show that ARMD simulations may offer important speedups on such processes. Our model system contains three types of particles, namely 1) nanodroplet, 2) target slab and 3) boundary particles. The nanodroplet consists of 2891 identical particles that are spherically distributed in a hexagonal close-packed arrangement (blue particles shown in Figure 3.7); the target slab consist of 344,988 identical atoms on the fcc lattice (grey and red particles shown in Figure 3.7); the boundary of the slab remains fixed and has no velocity (green particles shown in Figure 3.7). Interactions among particles were computed using the Lennard-jones potential. All simulations were performed using a 1 fs integration time step and ran for 75 ps. The initial velocity of the nanodroplet was set to 4 km/s (in negative z direction). Due to the high velocity of the nanodroplet, the neighbor list was constructed every fifth time step.

In order to measure the amorphization process, we observed the radial distribution function (RDF) of the impact volume of the target slab (red particles in Figure 3.7) for different sets of AR parameters, and compared the obtained RDFs with the one obtained with a reference MD simulation. In this benchmark, most slab particles were initially restrained, and gradually started to switch to an active state after impact.

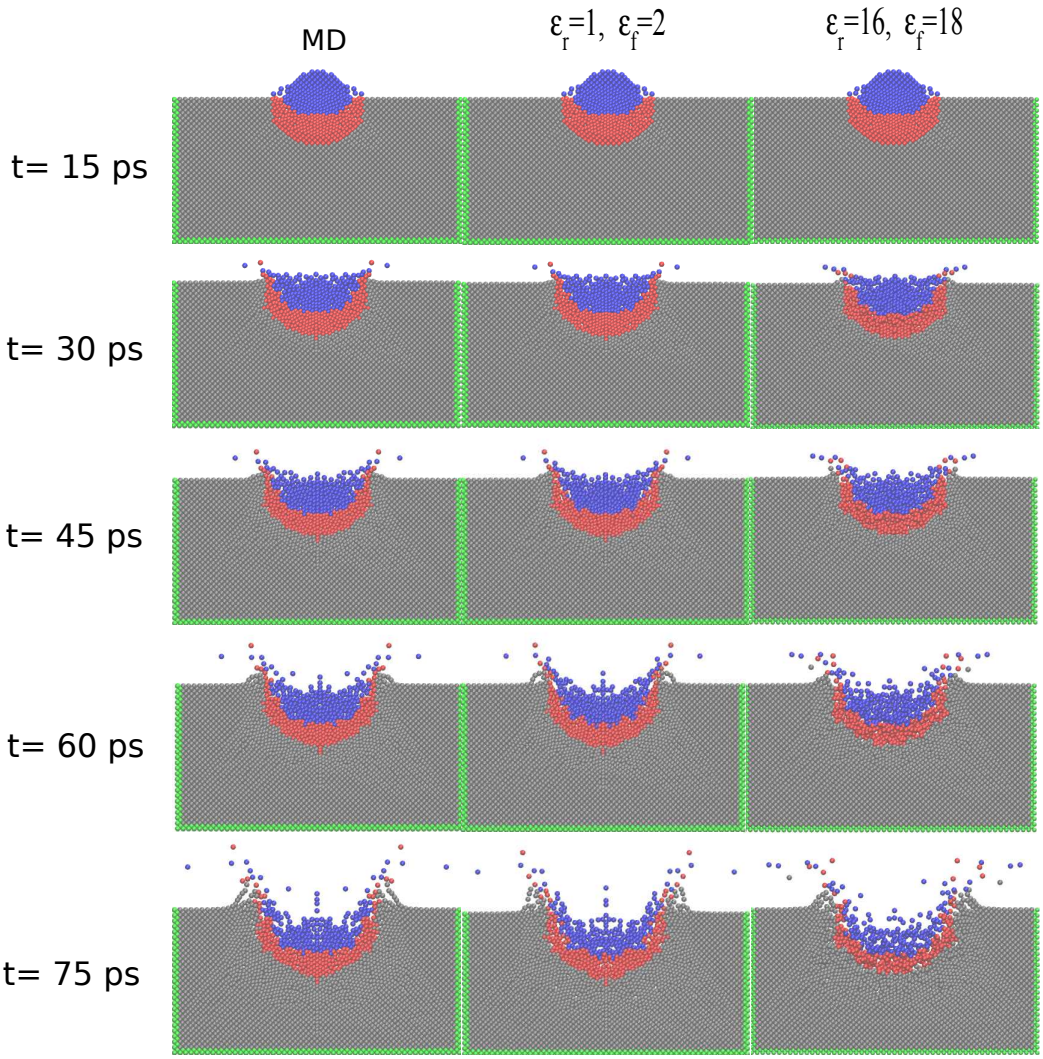


Fig. 3.7 Cross-section view of the nanodroplet impact at 0, 15, 30, 45, 60 and 75 *ps*. Red particles belong to the impact area. Green particles represent the fixed boundary of the impact slab.

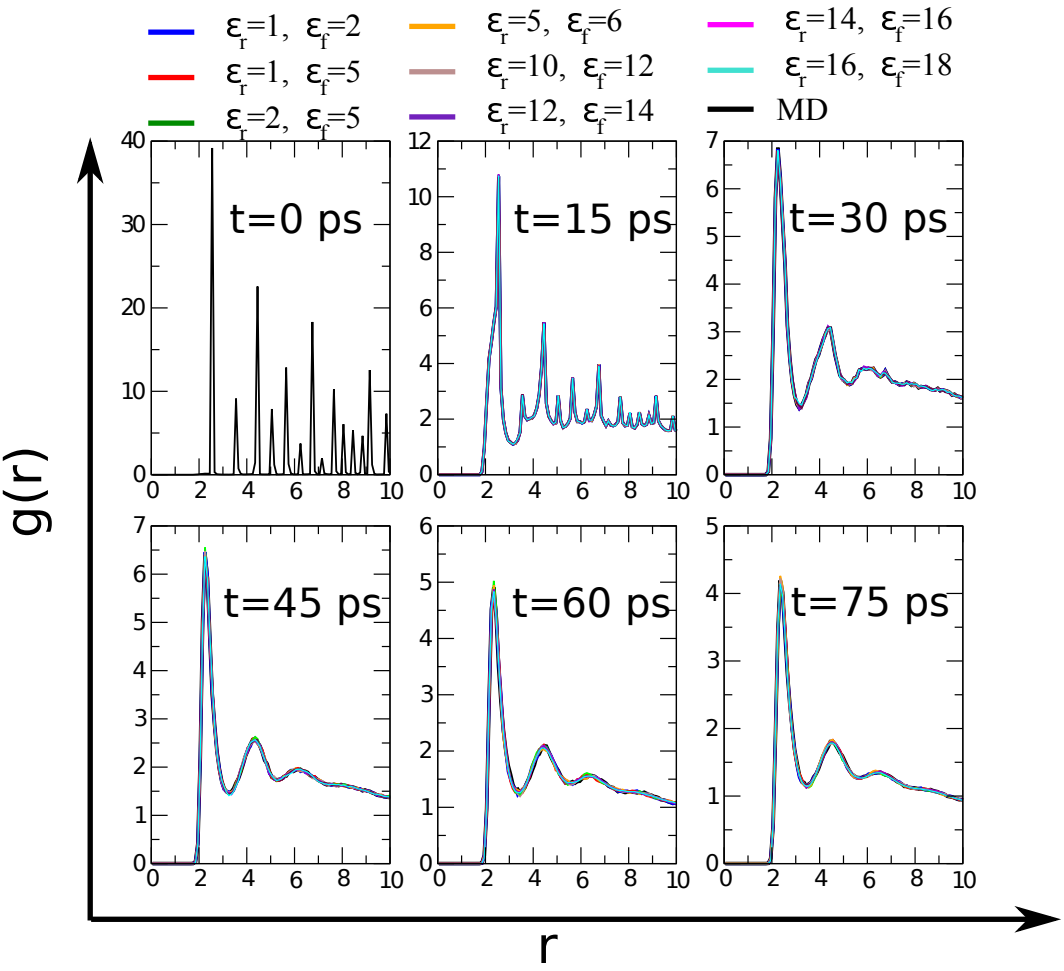


Fig. 3.8 Comparison of the RDFs of the impact area obtained with MD and ARMD. The RDFs obtained by ARMD with different AR parameters at different time-steps mostly coincide with the RDFs obtained by MD.

As shown in Figure 3.7, the crystalline structure at the impact area completely changes to a non-crystalline structure (amorphous state). This figure also shows that ARMD simulations allow us to trade between speed and accuracy. Lower values of AR parameters produce a trajectory similar to the MD trajectory, but higher values of AR parameters produce higher speedups. Figure 3.8 shows the obtained RDFs of the impact area (particles denoted as red in figure 3.7) at different time-steps. At the beginning of the simulation (at $t = 0ps$), the impact area had a crystalline structure. As the simulation proceeds, this crystalline structure started to deform and changed to an amorphous state (RDFs at $t = 15, 30, 45, 60$ and $75ps$). During the amorphization process, the RDFs obtained from ARMD simulations mostly coincide with the RDF obtained using MD (Figure 3.8). The RDFs of the impact area at $75ps$ obtained using both ARMD and MD had their first peak at 2.35Å and second peak at 4.5Å. In order to measure the speed obtained by ARMD, we ran each simulation 50 times and computed the

$\varepsilon_r(eV)$	$\varepsilon_f(eV)$	$\% < N_R >$	Speed up
1	2	56.4	2.3
1	5	60.79	2.9
2	5	61.87	3.1
5	6	66.49	4.25
10	12	72.49	4.7
12	14	75.53	5.1
14	16	79.34	5.9
16	18	84.21	6.82

Table 3.3 Table shows the AR parameters used for performing ARMD simulation of the hyper velocity impact.

average time spent. We measured the speedup with respect to the reference MD simulation. ARMD achieved 2.3-7X speedup with $56-85\%$ restrained particles as compared to MD. Table 3.3 shows the speedup obtained with different AR parameters. This benchmark shows that ARMD saves wall-clock time while obtaining the structural properties of a specific part of the system much faster than classical MD, thanks to the automatic particle state switching resulting from the AR Hamiltonian [79].

3.4.4 A single polymer chain in solution

The system was initially minimized and then equilibrated for 10000 steps. After the equilibration period, the system was simulated for 10^9 steps in the NVT ensemble. Initial velocities were assigned using Maxwell-Boltzmann distribution at temperature $k_BT = 1.2$.

One of the main goal of MD is to compute the statistical properties of the system by calculating averages. Averages obtained by MD are time averages and, if simulation is long enough to be converged, these time averages are equal to ensemble averages (ergodic hypothesis). However, averages over a trajectory are subject to two types of errors: systematic bias and statistical errors. Systematic bias is due to the use of a discrete time step, and statistical errors occur due to the quality of sampling (and may be large if averages are obtained from an undersampled or a short-trajectory).

Statistical errors in time averages may be estimated by measuring the variance of an observable l. The variance can be measured either by correlation time analysis or by a block-averaging scheme [75–78]. In previous studies of ARMD, time correlation analysis was used to measure errors and variance [80], and we use the same approach for error estimations of ARMD trajectories in the present paper. The correlation time τ_l of an observable l is the simulation time required for a trajectory to de-correlate its value from an initial value l_0.

Therefore, the correlation time for an observable provides an estimation of N_l^{ind}, the number of statistically independent values of the observable present in the trajectory: if t_{sim} denotes the simulation length of a trajectory, then $N_l^{ind} \sim t_{sim}/\tau_l$. Larger values of N_l^{ind} suggests good sampling for the given observable. The time τ_l depends upon the nature of observable[95].

In ARMD, restraining positions of particles introduces additional correlation in the system, thus yielding a larger correlation time for a given observable, as compared to MD ($\tau_l^{ARMD} \geq \tau_l^{MD}$). If ARMD simulations have the same length as MD simulations, then statistical errors are larger:

$$\tau_l^{ARMD} \geq \tau_l^{MD} \text{ and } t_{sim}^{ARMD} = t_{sim}^{MD} \implies N_{l_{ARMD}}^{ind} \leq N_{l_{MD}}^{ind}$$

Fortunately, statistical errors of averages obtained by ARMD may be reduced, and more statistically independent values may be obtained, by performing simulations with longer lengths (more time steps), since each time step costs less in wall-clock time. Therefore, the overall speedup in the NVT ensemble is a function of the speedup obtained in wall-clock time at each time step, and the time required to attain a given precision in the estimation of a given observable [80]. This speedup can be expressed as:

$$S = S_{algo} \frac{\sigma_{MD}^2}{\sigma_{ARMD}^2} \tag{3.10}$$

where S_{algo} represents the computational speedup at each time step adaptive algorithms (ANLs and single-pass incremental force update algorithms), σ_{MD}^2 is the variance of a given observable when using MD, and σ_{ARMD}^2 is the variance of the same observable when using ARMD. In this benchmark of a polymer in solvent, we chose the end-to-end distance of the polymer as an observable, and computed the correlation times with different AR parameters. Figure 3.11 shows the time correlation functions computed for end-to-end distances. As expected, the correlation function in the MD case is reduced in fewer time steps compared to the correlation functions in ARMD simulations. In wall-clock time, however, some ARMD trajectories decorrelate the end-to-end distance faster than MD, due to the reduction in the average cost of a time step.

Table 3.4 shows the averaged values of end-to-end distances and radius of gyration of the polymer chain. The averages obtained with MD and ARMD are approximately the same. This experimentally shows how, when the observable is converged, averages obtained with ARMD are the same as the ones obtained with MD (see [79] for a mathematical proof). Figure 3.10 shows the 2D projection of all trajectories as a function of end-to-end distance and radius of gyration of the polymer. Here as well we obtain unbiased position-dependent

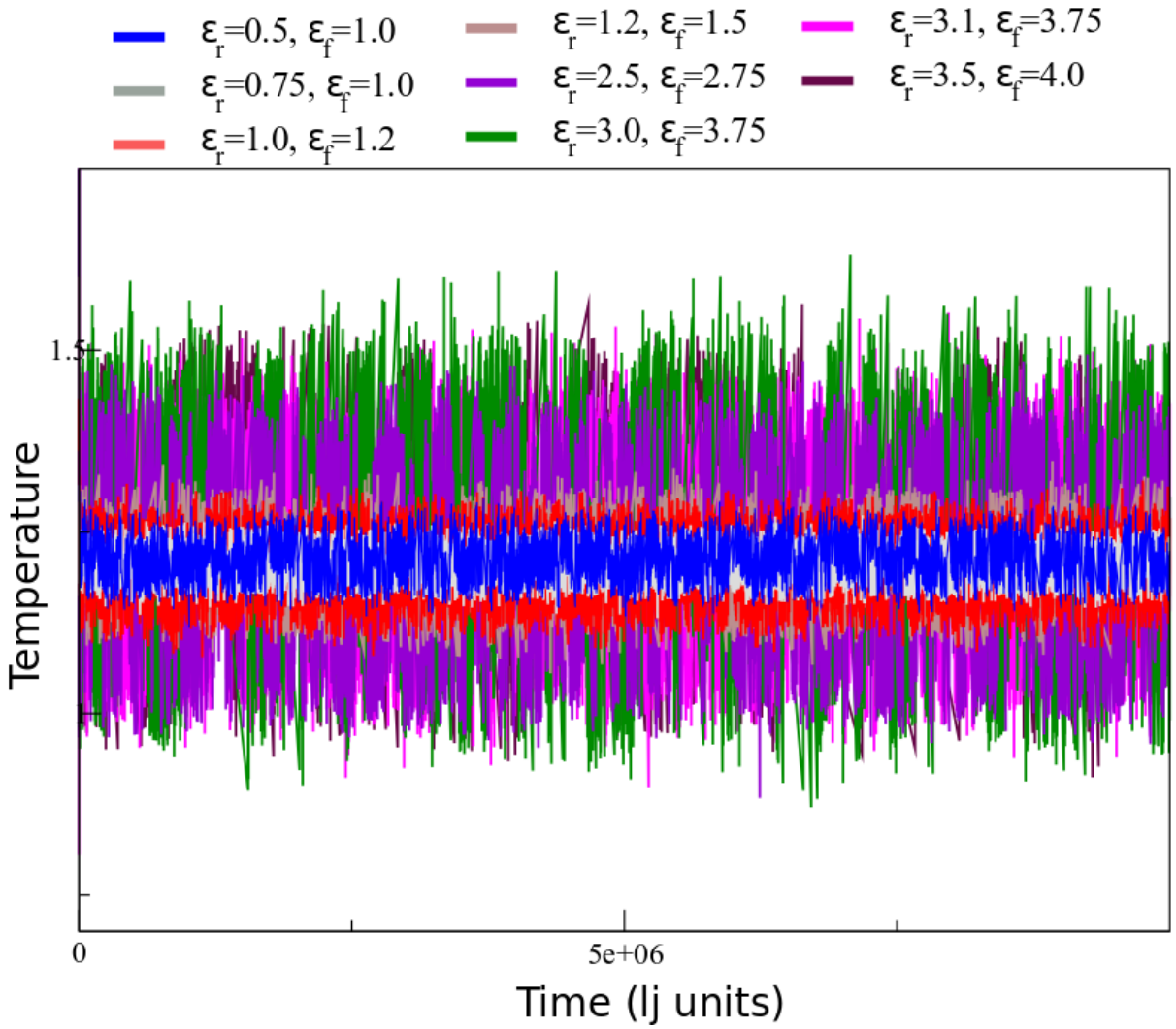


Fig. 3.9 Instantaneous temperature of the system with different AR parameters.

	MD	$\varepsilon_r/\varepsilon_f$							
		.5/1	.75/1.0	1.0/1.2	1.2/1.5	2.5/2.75	3.0/3.75	3.1/3.75	3.5/4.0
$\% < N_R >$	0	18	31	43	51	77.8	83	86	90
$R =< R^2 >^{\frac{1}{2}}$	8.4404	8.481	8.634	8.3013	8.368	8.394	8.569	8.3919	8.556
$R_G =< R_G^2 >^{\frac{1}{2}}$	3.523	3.547	3.585	3.520	3.55	3.535	3.546	3.534	3.569

Table 3.4 Summary of statistical properties obtained by MD and ARMD for the polymer benchmark. R is the end-to-end distance of the chain and R_G is the radius of gyration. Statistical averages obtained from ARMD and MD are similar.

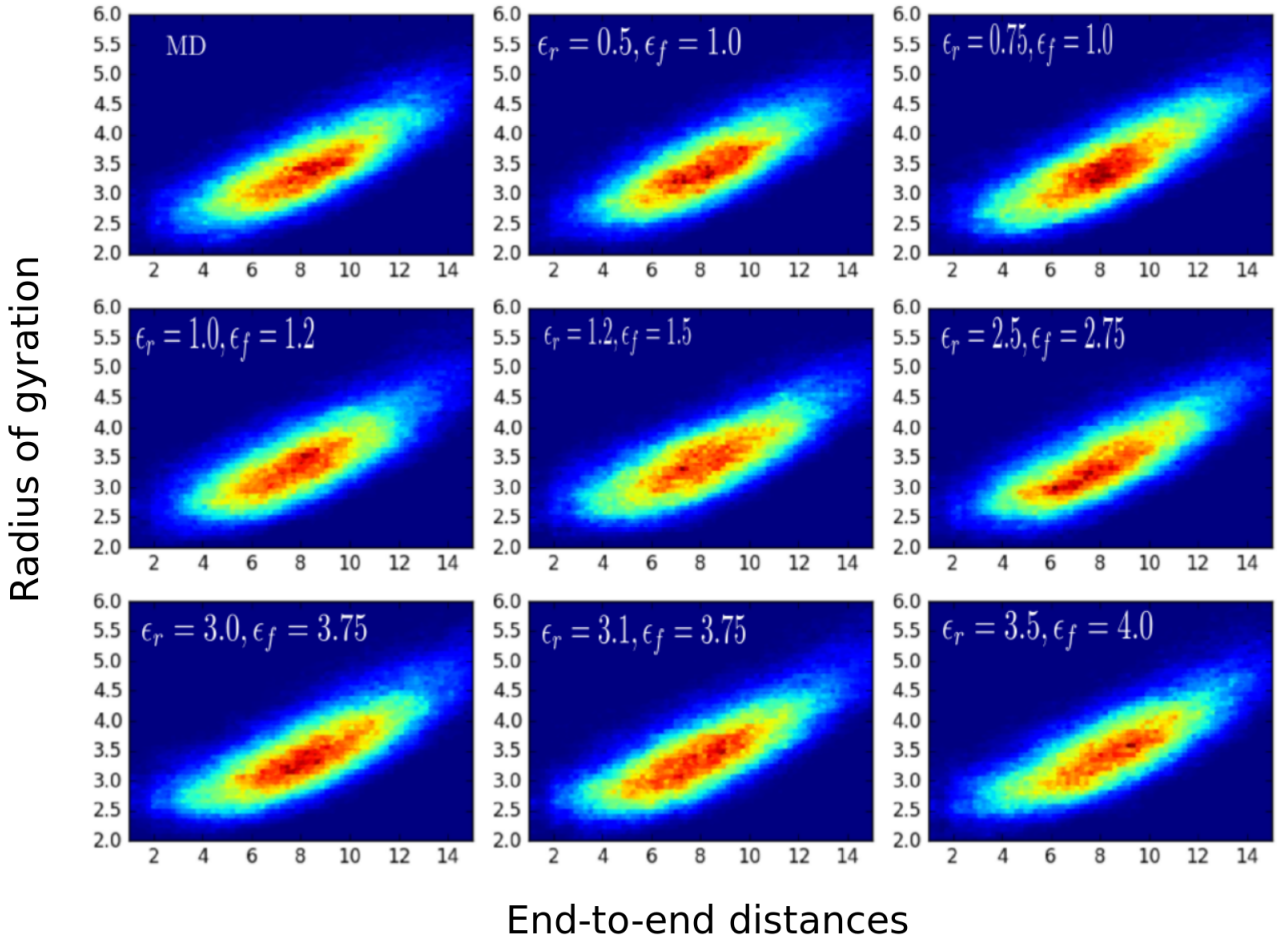


Fig. 3.10 Projection of trajectories on the end-to-end distance (R) and radius of gyration (R_g) of the polymer (colors represent the number of conformations in each bin). This figure illustrates that ARMD simulations produce unbiased positional statistics.

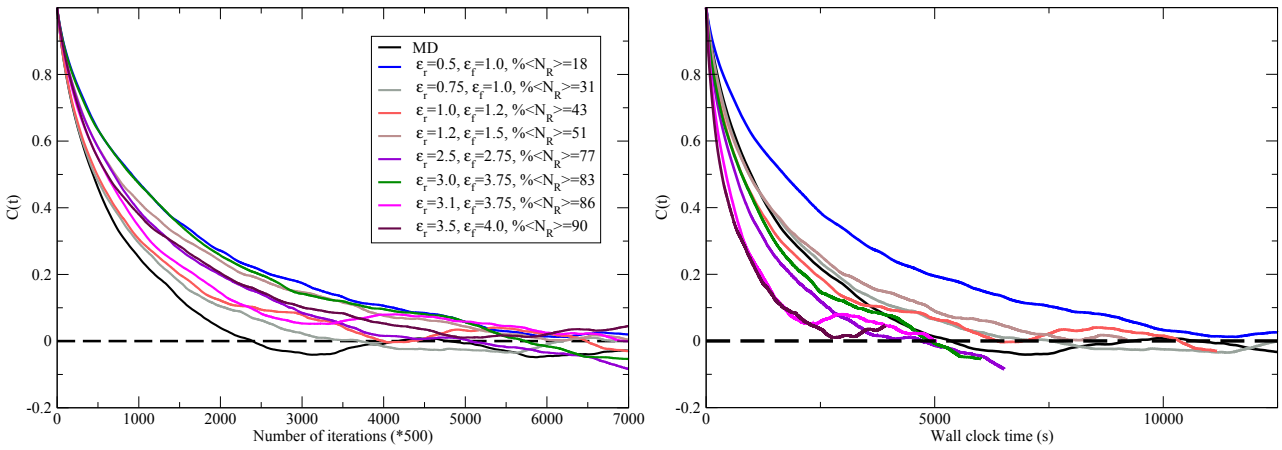


Fig. 3.11 Time correlation function of the end-to-end distance of the polymer. The left part shows that MD takes fewer iterations to decorrelate the end-to-end distance when compared to ARMD. The right part shows that, in wall-clock time, however, some ARMD simulations converge up to twice faster than a MD simulation.

averages. This illustrates that ARMD samples the same conformational space as the MD in the NVT ensemble[79].

3.5 Conclusions

We have presented a novel single-pass force update algorithm to speed up ARMD simulations. Unlike our previous two-pass algorithms, the new algorithm may result in a speedup even when a small percentage of particles is restrained. We have validated the approach on several benchmarks, and have shown that the single-pass algorithm may be applied to computing electrostatic interactions with the Wolf method. We showed how ARMD may be used to converge a given observable faster than with MD.

Chapter 4

Parallel Algorithms for Adaptively Restrained Molecular Dynamics

Abstract

Force computations are one of the most time consuming part in performing Molecular Dynamics (MD) simulations. Adaptively Restrained Molecular Dynamics (ARMD) makes it possible to perform fewer force calculations by adaptively restraining particles positions. This chapter introduces parallel algorithms for single-pass incremental force computations to take advantage of adaptive restraints using the Message Passage Interface (MPI) standard. The proposed algorithms are implemented and validated in LAMMPS, however, these algorithms can be applied to other MD simulators. We compared our algorithms with LAMMPS for performance and scalability measurements.

4.1 Introduction

Molecular Dynamics (MD) used to investigate the statistical and dynamic characteristic of complex systems. These complex systems are generally contain many atoms, simulating such complex systems to obtain experimental statistics needs hugs amount of computational time. The computational time associated with the complex system can be reduced by utilizing parallel abilities of MD cods. In general, MD codes are not memory intensive and paralleling the MD cods are not tedious. As mentioned in previous chapters, MD codes contain repetition of these steps:

1. Update Momenta
2. Update position
3. force calculation.

All three steps are parallelizable on parallel architectures. Parallel architectures are generally categories into two categories: Shared memory architecture and Distribiuted memory architecture

In this chapter, we will mainly focus on the algorithms for ARMD on distributed memory architectures. As shared memory architecture does not require any additional algorithm for

ARMD and work (updating positions momenta and computing forces) can be distributed on different threads and barrier can be applied before and after the force calculations. However, on distributed memories, new algorithms are needed to reduced the interprocessor communications and distribute the work among the processors. Communications among distributed memory is obtained by Message Passing Interface (MPI). MPI creates and distributes processes on different processor (from now on, we will use MPI terminology and refer processors as process).

In MD, a system containing N particles can be divided on P processes based on any of these three algorithm.

1. Atom-Decomposition Algorithm: all particles are distributed to MPI processes
2. Force decomposition Algorithm: Force matrix is distributed over the MPI processes.
3. Spatial-Decomposition Algorithm: mostly used in modern simulator and this algorithm is described below.

Spatial-Decomposition (SD) Algorithm

In spatial decomposition algorithm, a simulation box is subdivided P smaller sub-boxes such that all processes have one box. Particles on one process can interact with the particles on neighbor processors , these particles are categorised as border particles. In order to compute forces, particles needs to know the positions of its particles and neighboring processes particles. This communication is further improved by dividing sub-boxes into bins (based on cut-off) and particles belonging to the neighoring bins needs to exchange the particles position informations. The particles on one process that has to be communicated are refereed as border particles. Border particles on a processor are generally less in number as compared to the number of particles belong to the same processors. Communications with adjacent processes are obtained with east/west, north/south and up/down scheme. The modified version of SD algorithm that LAMMPS uses, involve forward communication of border particles position and reverse communication of forces on border particles. Figure 4.1 depicts a schematic diagram of this algorithm. The modified SD algorithm utilizes newton's third and communication among the processes are local in nature.

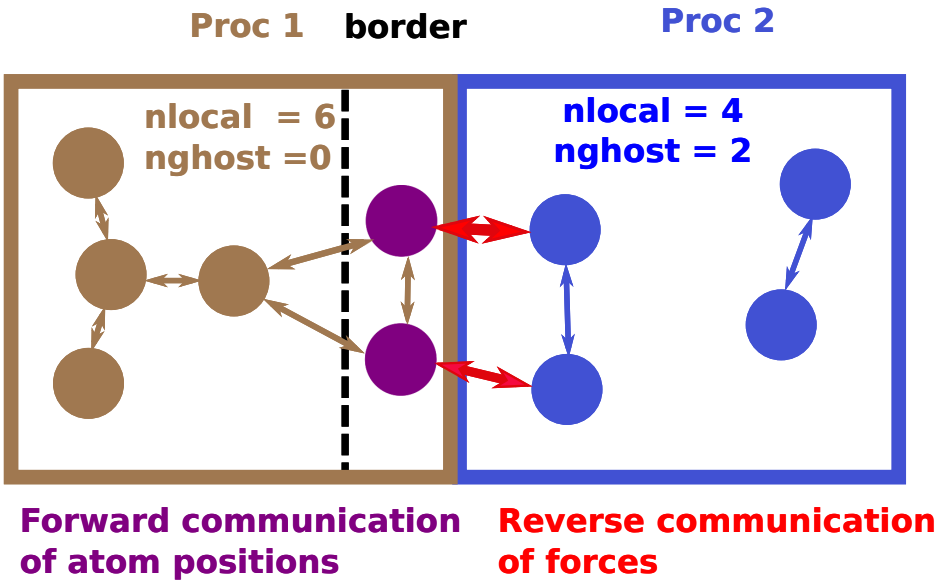


Fig. 4.1 Forward and backward communications in the LAMMPS.

A process contains two types of particles 1) local 2) ghost particles. Local particles belong to the current processor and ghost particles are on this processor due to communication routines. A processor is responsible for updating positions and momenta of local particles and computing forces of local as well as ghost particles. Performing simulations of N particles over P MPI processes contain repetition of steps mentioned in algorithm 14 (assuming that particles are distributed among processes using SD algorithm).

Algorithm 14: LAMMPS integration step

```
1 Update Momenta
2 Update Position
3 if (UpdateNeeded) then
4     Build Neighbour List for local particles
5 end
6 Forward Communications of border particles
7 Force Computations
8 Reverse Communications of forces
```

Communications between MPI processes take place in two steps: 1) Before forces computations on particles, positions of the border particles are communicated in the east, north and up directions. This communication in LAMMPS is called forward communication of the positions. 2) On the contrary to the forward communications, reverse communication send forces of the ghost particles in west, south and down directions. Due to the forward and reverse communications of positions and forces, force computations can be performed using

newton's third law. To compute forces correctly, updated positions and forces corresponding to the ghost particles need to be communicated in forward and reverse communications.

4.1.1 LAMMPS

LAMMPS stands for Large-scale Atomic/Molecular Massively Parallel Simulator and is a highly parallel and modular MD package. Parallel implementation of LAMMPS with MPI uses a Spatial-Decomposition (SD) techniques to partition the simulation domain into sub-domains and assigning each of them to one MPI process.

Force acting on a particle depends upon its neighboring particles, which may include particles belonging to neighboring sub-domains, therefore inter-processor communications are required for force computations. We will refer to particles whose positions needs to be communicated between MPI processes as border particles. In order to optimize border particles communication, each sub-domain is further partitioned into bins (based on cut-off radius), and then border particles are defined as particles belonging to the bins neighboring other sub-domains. Therefore, an MPI process contains two kind of particles: local particles (belonging to this sub-domain) and ghost particles (border particles from heighboring sub-domains). An MPI process is responsible for updating momenta and positions of its local particles, and computation of interactions between local particles (*local interactions*) and interactions of local particles with ghost particles (*ghost interactions*). LAMMPS performs two-way communication: positions are communicated in forward directions (in the east, north and up directions) and computed forces are communicated back in reverse directions (west, south and down). Due to this, LAMMPS can fully utilize the Newton's third law. After setting up the simulation, LAMMPS repeats the steps listed in Alg. 14.

In MD, forces are computed on all particles. In ARMD, in contrast, involves force computations based on only active interactions. In previous chapters, we proposed the ANL and incremental force update algorithms to simulate a system using ARMD. In fact, we proposed efficiently construction of the ANLs and single pass algorithm for incremental force update. These algorithms are efficient to simulate a system on single processor (or even shared memory architectures), on distributed system, these algorithms do not have any notion of ghost or local particles. Previous proposed algorithm for ARMD can also be used with MPI by introducing the notion of local and ghost particle. The following steps will perform

ARMD simulation of a system by using algorithms mentioned in the previous chapter:

Algorithm 15: Performing ARMD with MPI

1 Update momenta
2 Forward communications of border switched particles.
3 Incremental force calculations of switched particles.
4 Reverse communication of the forces due to the switched particles.
5 Update positions of local active particles.
6 Forward communications of positions of the boarder active particles.
7 Compute force
8 Reverse communications of ghost forces.

In order to perform ARMD using MPI requires two extra communication and barrier routines. First forward communication sends the position of switched particles to neighbouring processors, and force components due to switched particles are communicated using reverse communication. These extra barrier and communication routines reduce the number of force computations, however, limit the speed-up of a ARMD simulation. As communication and barrier routines act as a speed-breaker in achieving speed-up. In this chapter, we will propose new parallel algorithms to lower the communications and to reduce number of force computations.

4.2 Parallel ARMD Algorithms

In our work, we are taking advantage of the modular structure of LAMMPS: we implement ARMD as a separate module that allows usage of most implemented force-fields without any modifications. As mentioned above, in LAMMPS, each MPI process manages both local and ghost particles. In ARMD, local particles are further subdivided into active and restrained particles based on their instantaneous kinetic energy. Therefore, local interactions can be categorized as

- *active interactions*: interactions involving at least one active particle;
- *restrained interactions*: interactions involving restrained particles only.

Ghost interactions might also be subdivided into active and restrained interactions. However, this would require two extra communication and barrier routines for forward and reverse communications. While such a subdivision may reduce the number of force computations, additional communications may limit the speed-up achievable by ARMD simulations. Hence,

we propose a parallel algorithm in which only local interactions are subdivided into active and restrained interactions.

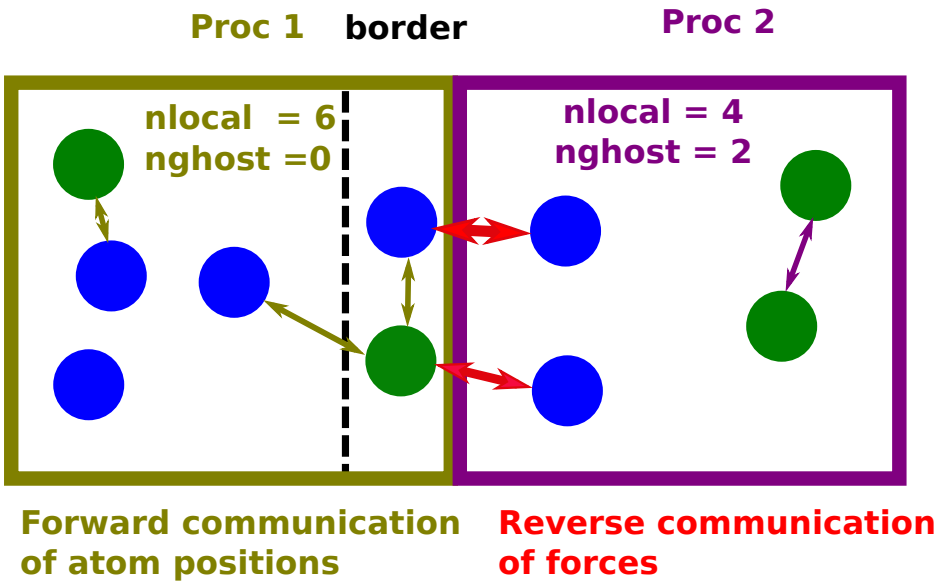


Fig. 4.2 A system containing 10 particles is divided between two processes (Proc 1 and Proc 2). Proc 1 sends positions of its two border particles (separated by a dashed line) by doing a forward communication. After forward communication is complete, Proc 2 has four local and two ghost particles, and Proc 1 has six local particles. Green (resp. blue) particles represent active (resp. restrained) particles. Each process computes forces acting on its local and ghost particles (forces are shown by arrows), while avoiding computations between restrained-restrained particles. After computing forces, forces acting on ghost particles (red arrows) are communicated back through reverse communication.

ARMD speeds up simulations due to its ability to incrementally update forces, instead of re-computing all of them. This is achieved through the use of single-pass incremental force update algorithm and Active Neighbor Lists (ANLs).

Algorithm 16: ARMD integration step

```
1  Update Momenta
2  Update A, R, S_A, S_R
3  if (UpdateNeeded) then
4      Build ANLs of local particles
5  end
6  SwitchedForces()
7  Update Positions
8  Forward communications of border particles
9  IncrementalForceComputation()
10 Reverse communications of forces
```

In order to efficiently parallelize ARMD in LAMMPS we introduce three main modifications (Alg. 16) to the LAMMPS integration step (Alg. 14):

1. Lists of active (A), restrained (R), switched to active (S_A) and switched to restrained (S_R) particles are updated at each timestep (line 2 in Alg. 16). ANLs are constructed when necessary (line 4 in Alg. 16) instead of the LAMMPS neighbor lists.

2. Particles that switched their state (from active to restrained or vice-versa) are taken care of (line 6 in Alg. 16).

3. Finally, force increments are computed (line 9 in Alg. 16) instead of recomputing all forces.

In the next sections, we present these algorithms for parallel ARMD.

4.2.1 MPI-enabled Active Neighbor List

This section introduces an algorithm to construct a MPI-enabled ANL that allows us to avoid calculating forces due to restrained interactions, and exploits Newton's third law. The MPI-enabled ANL provides an efficient way to compute forces involving active and ghost interactions. In order to use MPI functionality of LAMMPS (spatial decomposition and communication algorithms), we construct ANLs using cell and Verlet neighbor list algorithms (which are also used to build LAMMPS neighbor lists) [59, 60]. The ANL construction algorithm is shown in Alg. 17, where G is the list of ghost particles for the MPI process

managing particle i, and $NeighboringCells[i]$ is the list of boxes neighboring or containing particle i (27 boxes in 3D). The ANL of an active particle contains both active and ghost interactions, whereas the ANL of a restrained particle only contains ghost interactions. To allow the usage of Newton's third law, the ANL of particle i does not contain particle j if the ANL of particle j already contains particle i. Therefore, the ANL of i stores pair i–j in two cases (Alg. 17): 1) if particle i is active and particle j is either restrained or a ghost particle; 2) if particle i is restrained and particle j is a ghost particle.

Algorithm 17: $BuildANL(i)$

```
1 ANL(i) ← ∅
2 for j ∈ NeighboringCells[i] do
3     if (i ∈ A and (j ∈ R or i ∉ ANL(j)) or j ∈ G then
4         ANL(i) ← j ∪ ANL(i)
5     end
6 end
```

4.2.2 MPI-enabled Single-Pass Incremental Force Update Algorithm

In parallel ARMD, an MPI process contains four force components based on local and ghost interactions: F_{AA}, F_{AR} and F_{RR} due to active-active, active-restrained and restrained-restrained pairs, respectively, and F_{ghost} due to ghost interactions. The force acting on an active particle i can be expressed as $F_i = F_{AA} + F_{AR} + F_{ghost}$, and the force acting on a restrained particle i is $F_i = F_{AR} + F_{RR} + F_{ghost}$. The F_{AA} and F_{AR} force components are associated with active interactions, thus they need to be computed at each timestep. Forces due to ghost particles are computed irrespective of their state also at each timestep (Fig. 4.2). Although it is possible to further reduce force computations by using the state of ghost particles, it requires at least two more communications among MPI processes. Since distances between restrained particles remain unchanged, the force component based on these distances (F_{RR}) on each MPI process need to be computed only once (at the beginning of the simulation and when a particle switches from active to restrained) and can be retained as long as the involved particles are restrained. Therefore, at each timestep, an MPI process is responsible for locally computing F_{AA}, F_{AR} and F_{ghost} force components, and then for communicating the F_{ghost} force components to neighboring MPI processes (and use Newton's third law). The algorithm that incrementally updates forces is shown in Alg. 18, where C is a list of all particles on

an MPI process, and $\mathbf{f}_i^+$ and $\mathbf{f}_i^-$ store F_{RR} and F_{AR} force components, respectively. The total force acting on particle i is stored in $\mathbf{f}$.

Algorithm 18: *IncrementalForceComputation*()

```
1  for i ∈ C do
2      if i ∈ A or i ∈ G then
3          f_i^+ ← 0
4      end
5      else
6          f_i^+ ← f_i^+ − f_i^−
7      end
8      f_i^− ← 0
9  end
10 f ← f^+ + ComputeForces(A, ANL)
```

4.2.3 Switching states

In ARMD, at each time step, particles can switch states. Precisely, a particle can either gain enough momenta to become active (to have full dynamics or transition dynamics), or lose enough momenta to become restrained. We refer to these particles as switched particles.

When a particle switches from a restrained state to an active state, we update its ANL using algorithm 17. If a particle switches to a restrained state, we remove active interactions from its ANL.

Since we do not subdivide ghost particles into active and restrained particles, whenever a local particle switches its force components should be computed based on local interactions only, without any communications between MPI processes. To achieve this, for each switched particle we extract from the ANL a reduced ANL (ANL$'$) which contains local interactions only, i.e. local restrained particles (see Alg. 19). Forces acting on switched particles are computed according to the direction of switching (see Alg. 20). If $i \in S_R$, then the force component F_{RR} for this particle is computed (force component F_{AA} and F_{AR} switches to F_{AR} and F_{RR}, respectively) and the F_{RR} force components for local restrained neighbors of this particle are updated. If $i \in S_A$, then the force component F_{RR} is set to zero for this particle and

the F_{RR} force components of its local restrained neighbors are updated accordingly. These operations are done on each MPI process separately and do not require any communications.

Algorithm 19: $ExtractANL'(i)$

```
1 ANL'(i) ← ∅
2 for j ∈ ANL(i) do
3     if j ∈ R and (i ∈ R or (i ∈ A and j ∉ S_A ∪ S_R)) then
4         ANL'(i) ← ANL'(i) ∪ j
5     end
6 end
```

Algorithm 20: $SwitchedForces()$

```
1 for i ∈ S_R do
2     ExtractANL'(i)
3     f_i^+ ← f_i^+ + ComputeForces(i, ANL'(i))
4 end
5 for i ∈ S_A do
6     BuildANL(i)
7     ExtractANL'(i)
8     f_i^- ← f_i^- + ComputeForces(i, ANL'(i))
9 end
```

The algorithm 16 performs ARMD simulations using multiprocessor; however, obtained speed-up will depend upon not only reduction in force calculation but also other factors as well. These factors are

- Distribution of work among MPI processes
- Work imbalance
- Communication among MPI processes

The performance of a simulation depends upon the distribution of particles on processors. The main reason for that is the work imbalance i.e. certain processes (more dense part of the system) will perform more work as compared to the other processes. In MD, a barrier in work is necessary after force computation. LAMMPS uses spatial domain decomposition algorithm to partition a simulation box containing N particles into P domains. Each domain then assigned to a MPI process. LAMMPS provides shift or RCB methods to perform load balancing.

Load balancing

MD codes perform re-balancing or repartitioning of a simulation box either at a regular frequency (generally after 1000 time-step) or whenever the imbalance exceed a certain threshold.

The work (W) associated to a domain can be a cost function based on that balancing needs to be performed. Most commonly used cost function is number of atoms on a domain. This cost function is optimum for a system containing only one type of particle and might lead to imbalance when two or more than two types of particles present in the system. For example, a simulation box containing two type of particles 1) nano-particles 2) solvent particles. Solvent particles generally interact with less expensive potential; however, nano particles interact with other nano particles via a complex and relatively expensive potentials. For these type of systems, cost function as number of particles might not be appropriate. Another choice of cost function can be number of average neighbors, that can be analytically determine by using density and volume of the simulation box. Complex potential can also be considered by multiplying with some weight for interacting particles. These cost function is already implemented in the new LAMMPS version. These broad and diverse choice of cost function is efficient as particles do not change their type; however, in ARMD a particle is either active or restrained and the state of this particle might change based on its instantaneous kinetic energy. Because of the adaptive nature of particles, ARMD can lead to a huge imbalance even for a homogeneous system containing one type of particles. One solution can be considering cost function as a number of active particles, but this might again lead to imbalance as particles start to switch their state. In ARMD, amount of work (the load) is not only proportional to the active particles but also related to the restrained particles. Figure 4.3 demonstrates the imbalance in the load for a system containing 9 particles. These 9 particles are distributed by considering that amount of work is only proportional to the number of active particles. We introduced a AR load balancing scheme that takes account of loads associated with active and restrained particles. Instead of distributing equal number of particles on each processor, AR load-balancer distribute equal weights on each processors. Weights are calculated by either adding or multiplying load associated with particle.

$Weight_P = \sum_{i=0}^{i=nlocal} load_i$

To optimize the load balancing approach used in LAMMPS for ARMD, we introduce an AR load balancing scheme that uses RCB weights of particles and takes both these interactions into account. Instead of assigning equal numbers of particles to each process, the AR load balancer distributes equal weights to each process as in RCB. To address this, we introduce two load factors α and β: α is a weight associated with active particles; β is

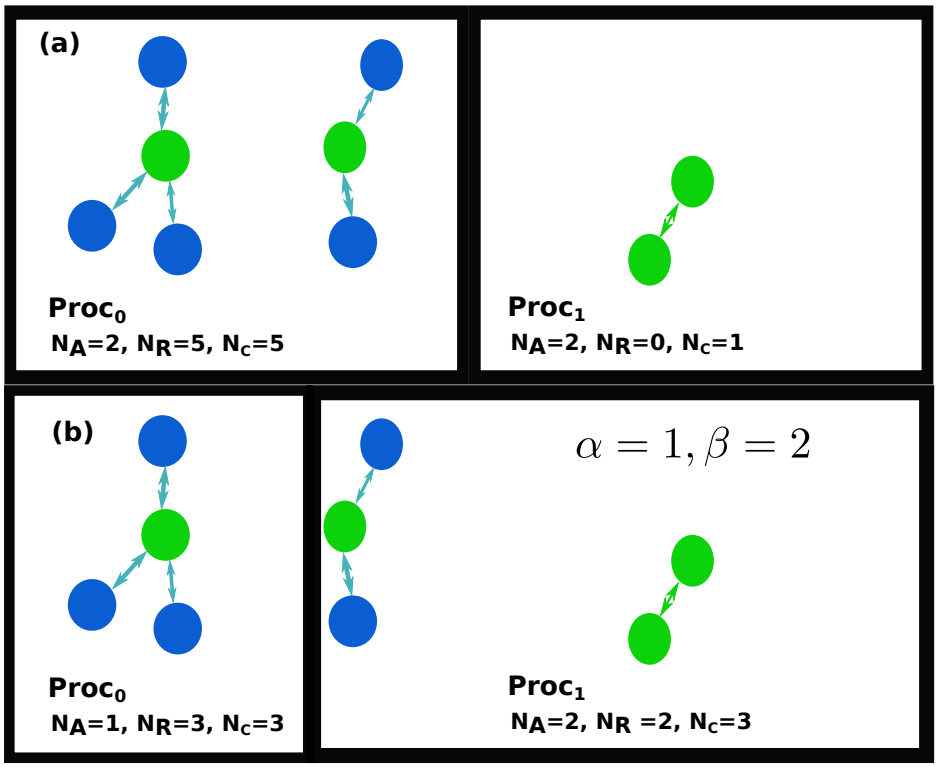


Fig. 4.3 A system containing 9 particles are partitioned into two domain by distributing the equal number of active particles. Despite having equal number of active particles (i.e. 2) on both domaim, this decomposition leads to load imbalance. Work in domain 1 is somehow related to the number of active interaction pairs present in the system (i.e. number of computation N_C). By using load as number of active particles, processor 0 perform 5 computations whereas processor 1 needs to perform only one computation. In AR load balancer

a weight associated with restrained particles. The weight for a process i can be expressed as $\alpha N_A^i + \beta N_R^i$, where N_A^i and N_R^i are number of active and restrained particles, respectively, assigned to i-th process. If values α and β are the same, then equal numbers of particles are distributed over processes.

The choice of parameters α and β is problem dependent and can be done in two ways: 1) user defined and constant; 2) adaptively updated starting from a user defined or default values. Since it is the ratio between α and β which guides the load balancing, parameter α can be set constant and equal to 1, while parameter β can be adaptively updated as

$$\beta = \frac{\sum_{i=0}^{p} t_R^i}{\sum_{i=0}^{p} t_A^i},$$

where the sum is done over all processes (p is the number of processes), t_A^i and t_R^i are time spent on routines associated with active and restrained particles, respectively, on i-th process. In our benchmarks, we use constant parameters $\alpha = 1$ and $\beta = 0.5$.

4.3 Results and Discussion

To show the performance and scalability of our MPI-enabled implementation of ARMD we present results for a standard Lennard–Jones (LJ) liquid benchmark [74].

The Lennard-Jones potential is often used to model and benchmark van der Waals forces in MD simulations. The potential is also referred to as 12-6 potential and is expressed as:

$$U = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

where ε is a well depth, σ and r are minimum-energy and instantaneous inter-particle distances, respectively. The Lennard-Jones potential is truncated at a cut-off distance $r_c = 2.5\sigma$; beyond this distance, the truncated potential is set to zero. Atoms are placed in a 3D cubic domain according to the fcc lattice, with a lattice constant equal to 0.8442; periodic boundary conditions are used; the initial temperature of the system is set to $T^* = 1.44$; timestep $\Delta t = 0.001$; the neighbor list is updated every 20 timesteps. All parameters are given in standard dimensionless LJ units. Simulations are done in the NVE ensemble for 5000 timesteps. The data on performance and the percentage of restrained particles are averaged over timesteps. All benchmarks are performed using a cluster with 8 nodes equipped with 8/16 CPUs Intel Xeon E5540 and a Gigabit Ethernet network. We run the benchmark both for ARMD and for LAMMPS on one node with different number of processes and on four nodes with four processes per each node (16 processes on total). The results obtained for ARMD are compared with LAMMPS performance results for the same systems computed on the same equipment. To compare performance, we use a standard performance metric of millions of atom-timesteps per second provided by LAMMPS.

The amount of interaction computations in ARMD depends on the percentage of restrained particles, and, therefore, the performance of ARMD will change based on this percentage. To test the performance of our MPI-enabled implementation of ARMD depending on the percentage of restrained particles, we simulate a system of 864 000 particles using different ARMD parameters $\varepsilon_r, \varepsilon_f$ to get different percentages of restrained particles. The comparison of results obtained with LAMMPS is shown in Fig. 4.4, with the averaged percentage of restrained particles on x axis. For low percentages of restrained particles, ARMD shows less performance in comparison with LAMMPS due to additional opera-

tions introduced in ARMD; ARMD starts to overperform LAMMPS when the percentage of restrained particles in the system reaches some threshold (break-even point). With an increasing number of MPI processes this break-even point shifts to a larger percentage of restrained particles. In this benchmark, ARMD outperforms LAMMPS if more than 60% of restrained particles is present in the system. This threshold depends on a number of factors: the cluster architecture; the simulated problem; the ε_r and ε_f parameters; the distribution of active particles over simulated domain, *e.g.* if all active particles are placed in one region of the domain then there will be fewer number of active–restrained interactions as compared to the case of even distribution of active particles over domain. In order to assess the worst case scenario — active particles are homogeneously distributed over the domain — we apply the same parameters $\varepsilon_r, \varepsilon_f$ for all particles in the simulated system. For non-homogeneous systems, ARMD will give better performance.

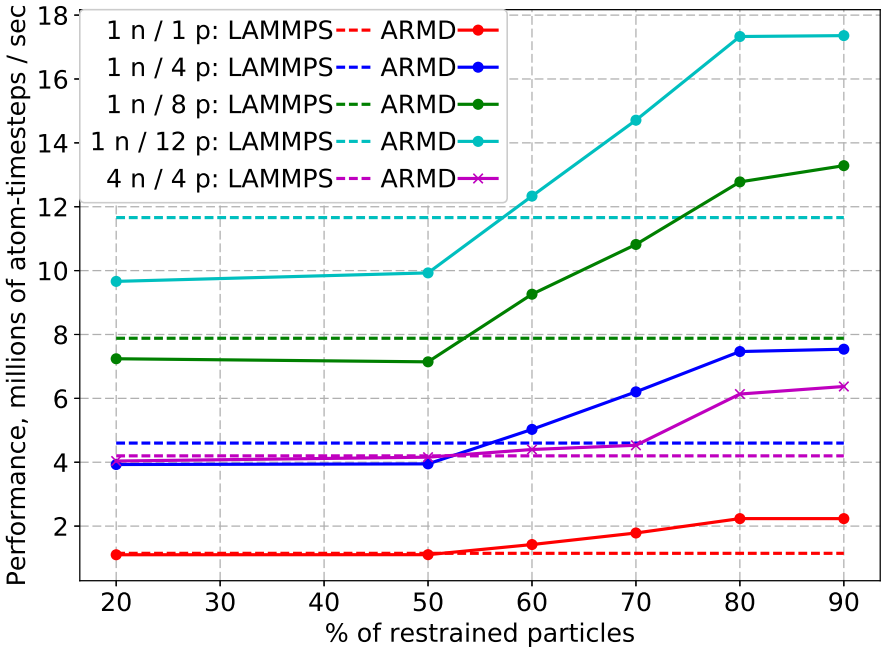
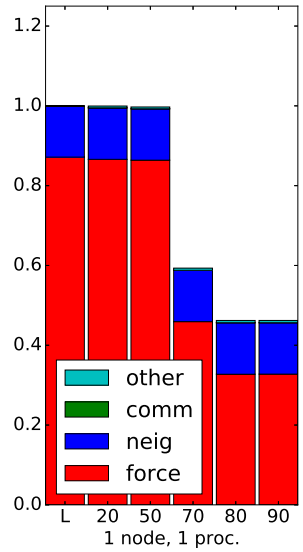
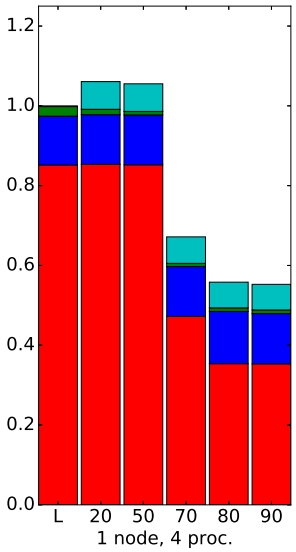


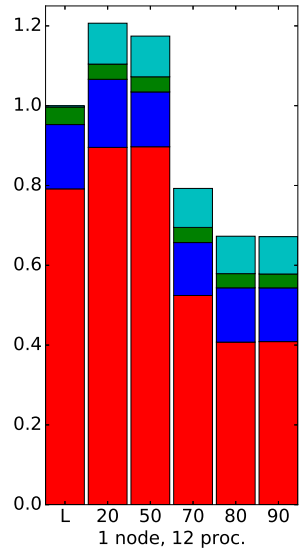
Fig. 4.4 Performance depending on the percentage of restrained particles for different number of nodes (n) and processes (p) per each node. Performance of non-modified LAMMPS is shown as a reference (dotted lines) — it does not depend on the percentage of restrained particles.



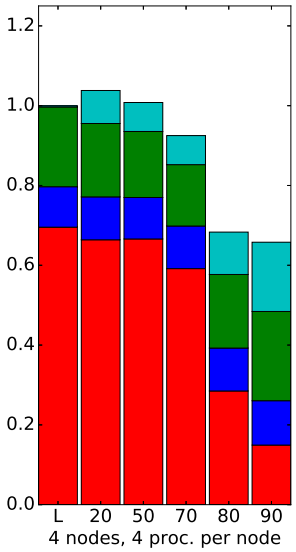
(a) 1 node, 1 process



(b) 1 node, 4 processes



(c) 1 node, 12 processes



(d) 4 nodes, 4 processes per node

Fig. 4.5 Breakdown of wall-clock time for 1 and 4 nodes with 4 processes per each node normalized by LAMMPS timing (L) for different percentage of restrained particles (20%, 50%, 70%, 80%, 90%). Other – Load balancing, ARMD routines for switched particles (ANL and force computations of switched particles), position & momenta update; comm – communications; neig – neighbor list construction; force – force computation.

Figure 4.5 shows a breakdown of wall-clock time normalized by LAMMPS time for different number of processes. With the increasing percentage of restrained particles, ARMD total time decreases due to a decrease in time spent on force computations. This decrease in time occurs after the aforementioned break-even point. If the percentage of restrained particles is smaller than this threshold, ARMD may even provide worse performance than LAMMPS (see Fig. 4.5b, 4.5c, 4.5d) because of additional computations due to switched particles (force computations, updating the ANL). An increase in the number of processes leads to an increase in time spent on ARMD routines due to load-balancing, updating the ANL of particles switched from restrained to active. Since it is necessary to take into account ghost particles while updating the ANL, the more processes are used the more ghost particles are present and should be included in the ANL.

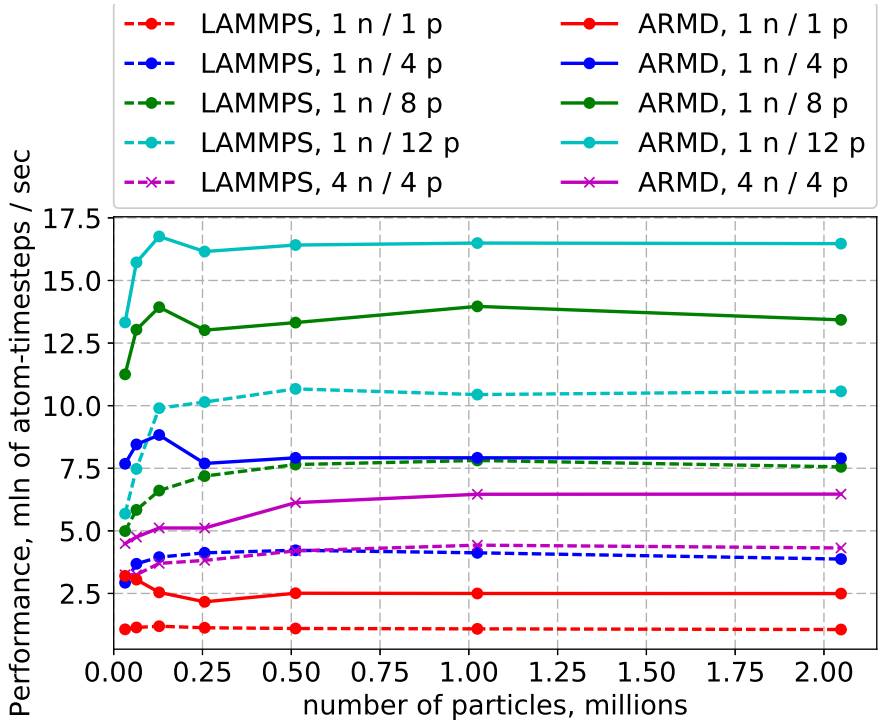


Fig. 4.6 Performance of ARMD and LAMMPS depending on the number of particles in the system for different number of nodes (n) and processes (p) per each node.

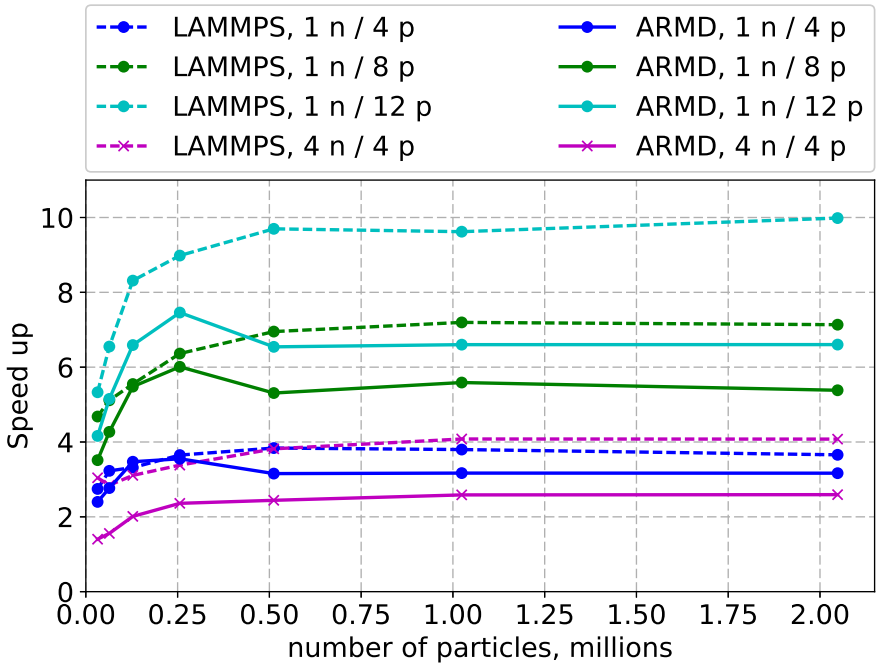


Fig. 4.7 Speed up of ARMD and LAMMPS for different number of nodes (n) and processes (p) per each node compared to their serial versions

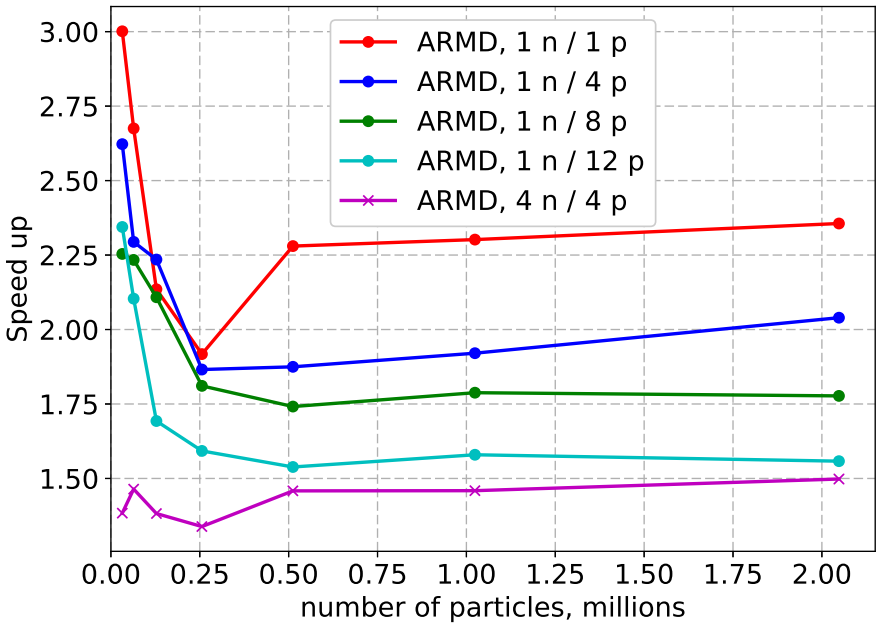


Fig. 4.8 Speed up of ARMD in comparison with LAMMPS for the same number of number of nodes (n) and processes (p) depending on the number of particles in the system

To show the scalability of our parallel implementation of ARMD depending on the number of particles in the simulated system, we fix parameters $\varepsilon_r, \varepsilon_f$ so there will be on average 80% of restrained particles at each timestep (since ARMD is most useful to apply when the percentage of restrained particles in the system is relatively high). ARMD and LAMMPS show good scaling of the performance with the number of particles (Fig. 4.6). As can be seen from Fig. 4.5, ARMD total time decreases due to a decrease in time spent on force computations, while time spent on the neighbor list construction and communications stays the same as compared to LAMMPS for the same configuration of nodes and processes. Since in the case of ARMD the force computation part — usually the most parallelizable part — takes less time, the speed up of ARMD compared to ARMD on one core is less than speed up of LAMMPS compared to serial LAMMPS (Fig. 4.7) due to the Amdahl's law. Speed up of ARMD in comparison with LAMMPS for the same configuration of nodes and processes also decreases with the number of used processes (Fig. 4.8) due to the same reason: the amount of interactions computations per process decreases, while time consumed by other operations common with LAMMPS, i.e. neighbor list construction and communications, stays the same. The fact that the performance of ARMD on 4 nodes is smaller than on 1 node (with 4 processes per node) (Fig. 4.6) is due to the significant amount of communications between nodes and a high-latency network.

4.4 Conclusion

ARMD allows us to accelerate MD computations by decreasing the number of interaction computations and provides better performance than classical MD for systems with a sufficiently large percentage of restrained particles (e.g. more than 60% for the benchmark used in the chapter). This threshold depends on the simulated problem and the architecture of a computational system. When the percentage of restrained particles in the simulated system becomes smaller than this threshold, computations may be switched from ARMD to classical MD, and returned to the usage of ARMD once this percentage reaches the threshold. The suggested paralellization of ARMD using MPI allows us to gain an additional speed up by the usage of multi-core CPUs and distributed systems. The ARMD shows good scalability with the number of particles in the simulated system. However, since ARMD accelerates only the interaction computation part of MD, the speed up of ARMD compared to LAMMPS for the same combination of nodes and processes decreases with the number of processes due to Amdahl's law. To overcome this limitation on distributed systems, we will investigate the possibility of developing a new approach that decreases the amount of data necessary to communicate by taking into account the state (active or restrained) of ghost particles. Also,

we now want to develop parallel ARMD algorithms for central or graphics processing units in combination with parallelization over a distributed system, in order to fully utilize modern clusters.

Chapter 5

Conclusion and future perspective

To conclude this dissertation, we now briefly describe the main contributions and provide potential directions for future research. We designed algorithms to simulate a system using adaptively restrained molecular dynamics in both the NVE and the NVT ensembles. In particular, we presented, for the first time, the active neighbor list algorithm, the single-pass incremental force computation algorithm and on-the-fly update of forces and neighbor lists of switching particles. These algorithms alleviate the need for all force computations, hence simulations can be performed faster. Unlike previously proposed two-pass algorithms for adaptive simulations, the new single-pass incremental force update algorithm reduces the number of computations by taking advantage of force decompositions, ANLs and switching particles. For a given wall-clock time, the single-pass algorithms perform more integration steps than two-pass algorithms.

These proposed algorithms have been integrated in the LAMMPS MD simulator and validated on diverse benchmarks in the NVE as well as in the NVT ensembles. In the NVE ensemble, AR parameters allow to tune between precision and speed up, and in the NVT ensemble, allow to obtain positional unbiased equllibrium statistical averages, and equilibrium structural properties of simulated systems. Statistical errors in estimated averages may be reduced by performing a longer ARMD simulation and by assigning appropriate AR parameters.

In the future, we would like to use our adaptive algorithms to explore ARMD in the following directions:

Initially, we would like to validate these algorithms on well-known biological benchmarks. It would be interesting to adaptively restrain the low-amplitude and high-frequency fluctuations present in a system, this might compel a biomolecule to attain the high-amplitude and low-frequency motions.

Next, we would like to use a combined approach by integrating ARMD with enhanced sampling methods like meta-dynamics, umbrella sampling and steered molecular dynamics in order to accelerate the phase space sampling. This integration might be useful in simulation of complex biological phenomena and in generating a free energy profile of a simulated system. This combined approach would also be interesting to understand the impact of solvent flexibility on a biomolecule. Precisely, adaptively restrained simulations of membrane proteins, while applying AR parameters on membranes alone, might give an insight into the effect of membrane fluidity on trans-membrane or peripheral proteins. Furthermore, ARMD simulations of ion-channels by adaptively restraining the membrane might be useful to understand the role of membrane flexibility on ion-channels.

Subsequently, in order to perform simulations of materials and electrolytes using ARMD, we would like to develop new adaptive algorithms for multi-body potentials and for long-range interactions. Our proposed algorithms can be used to compute forces based on a three-body potential; however, new algorithms are needed for potentials that involve many-body terms.

Thereafter, we are interested in exploring the possibility of using a larger time step in ARMD simulations. We believe that adaptively restraining the fast degrees of freedom in a system might allow us to use a bigger integration step during a simulation. We would like to use ARMD in conjunction with constraint algorithms like SHAKE or other methods like Hydrogen-Mass repartitions that might allow us to use a larger time step.

Following, we would like to reduce the number of computations further by performing an ARMD simulation with the multiple time-stepping algorithm.

Finally, we would like to investigate the affect of AR parameters on the convergence of an observable and the sampling of phase space.

References

[72] Abraham, M. J., Murtola, T., Schulz, R., Páll, S., Smith, J. C., Hess, B., and Lindahl, E. (2015). Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1:19–25.

[13] Amadei, A., Linssen, A. B. M., and Berendsen, H. J. C. (1993). Essential dynamics of proteins. *Proteins: Structure, Function, and Bioinformatics*, 17(4):412–425.

[91] Aragones, J. L., Sanz, E., Valeriani, C., and Vega, C. (2012). Calculation of the melting point of alkali halides by means of computer simulations. *The Journal of Chemical Physics*, 137(10):104507. 00013.

[79] Artemova, S. and Redon, S. (2012). Adaptively Restrained Particle Simulations. *Physical Review Letters*, 109(19):190201.

[90] Baker, J. A. and Hirst, J. D. (2014). Accelerating electrostatic pair methods on graphical processing units to study molecules in supercritical carbon dioxide. *Faraday Discuss.*, 169:343–357.

[39] Bally, T. and Rablen, P. R. (2011). Quantum-chemical simulation of 1h nmr spectra. 2.† comparison of dft-based procedures for computing proton–proton coupling constants in organic molecules. *The Journal of organic chemistry*, 76(12):4818–4830.

[36] Becke, A. D. (2014). Perspective: Fifty years of density-functional theory in chemical physics. *The Journal of chemical physics*, 140(18):18A301.

[4] Benkabou, F., Aourag, H., and Certier, M. (2000). Atomistic study of zinc-blende CdS, CdSe, ZnS, and ZnSe from molecular dynamics. *Materials Chemistry and Physics*, 66(1):10–16.

[43] Bennett, C. H. (1975). Mass tensor molecular dynamics. *Journal of Computational Physics*, 19(3):267–279.

[34] Berendsen, H. J. (2007). *Simulating the physical world: hierarchical modeling from quantum mechanics to fluid dynamics*. Cambridge University Press.

[82] Bosson, M., Grudinin, S., Bouju, X., and Redon, S. (2012). Interactive physically-based structural modeling of hydrocarbon systems. *Journal of Computational Physics*, 231(6):2581–2598.

[68] Bowers, K. J., Chow, E., Xu, H., Dror, R. O., Eastwood, M. P., Gregersen, B. A., Klepeis, J. L., Kolossvary, I., Moraes, M. A., Sacerdoti, F. D., et al. (2006). Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 84. ACM.

[83] Bransburg-Zabary, S., Nachliel, E., and Gutman, M. (2002). A Fast in Silico Simulation of Ion Flux through the Large-Pore Channel Proteins. *Biophysical Journal*, 83(6):3001–3011.

[69] Brooks, B. R., Brooks, C. L., MacKerell, A. D., Nilsson, L., Petrella, R. J., Roux, B., Won, Y., Archontis, G., Bartels, C., Boresch, S., et al. (2009). Charmm: the biomolecular simulation program. *Journal of computational chemistry*, 30(10):1545–1614.

[37] Buda, F. (2008). Density functional theory and car-parrinello molecular dynamics methods. *Biophysical Techniques in Photosynthesis*, pages 487–499.

[35] Car, R. and Parrinello, M. (1985). Unified approach for molecular dynamics and density-functional theory. *Physical review letters*, 55(22):2471.

[30] Chifotides, H. T. and Dunbar, K. R. (2005). Interactions of metal- metal-bonded antitumor active complexes with dna fragments and dna. *Accounts of chemical research*, 38(2):146–156.

[70] Christen, M., Hünenberger, P. H., Bakowies, D., Baron, R., Bürgi, R., Geerke, D. P., Heinz, T. N., Kastenholz, M. A., Kräutler, V., Oostenbrink, C., et al. (2005). The gromos software for biomolecular simulation: Gromos05. *Journal of computational chemistry*, 26(16):1719–1751.

[6] Curtin, W. A. and Miller, R. E. (2003). Atomistic/continuum coupling in computational materials science. *Modelling and Simulation in Materials Science and Engineering*, 11(3):R33.

[62] Darden, T., York, D., and Pedersen, L. (1993). Particle mesh Ewald: An N.log(N) method for Ewald sums in large systems. *The Journal of Chemical Physics*, 98(12):10089–10092.

[26] De Simone, A., Esposito, L., Pedone, C., and Vitagliano, L. (2008). Insights into stability and toxicity of amyloid-like oligomers by replica exchange molecular dynamics analyses. *Biophysical journal*, 95(4):1965–1973.

[55] Dodson, B. W. (1987). Development of a many-body tersoff-type potential for silicon. *Physical Review B*, 35(6):2795.

[15] Duan, Y. and Kollman, P. A. (1998). Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science*, 282(5389):740–744.

[10] Elber, R. and Karplus, M. (1987). Multiple conformational states of proteins: a molecular dynamics analysis of myoglobin. *Science*, 235(4786):318–321.

[46] Elstner, M., Frauenheim, T., and Suhai, S. (2003). An approximate dft method for qm/mm simulations of biological structures and processes. *Journal of Molecular Structure: THEOCHEM*, 632(1):29–41.

[63] Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. G. (1995). A smooth particle mesh ewald method. *The Journal of chemical physics*, 103(19):8577–8593.

[92] Fanourgakis, G. S. (2015). An Extension of Wolf's Method for the Treatment of Electrostatic Interactions: Application to Liquid Water and Aqueous Solutions. *The Journal of Physical Chemistry B*, 119(5):1974–1985.

[86] Fennell, C. J. and Gezelter, J. D. (2006). Is the Ewald summation still necessary? Pairwise alternatives to the accepted standard for long-range electrostatics. *The Journal of Chemical Physics*, 124(23):234104.

[61] Fomin, E. S. (2011). Consideration of data load time on modern processors for the Verlet table and linked-cell algorithms. *Journal of Computational Chemistry*, 32(7):1386–1399.

[19] Freddolino, P. L., Liu, F., Gruebele, M., and Schulten, K. (2008). Ten-microsecond molecular dynamics simulation of a fast-folding ww domain. *Biophysical journal*, 94(10):L75–L77.

[40] Freier, E., Wolf, S., and Gerwert, K. (2011). Proton transfer via a transient linear water-molecule chain in a membrane protein. *Proceedings of the National Academy of Sciences*, 108(28):11435–11439.

[28] Fulle, S., Christ, N. A., Kestner, E., and Gohlke, H. (2010). Hiv-1 tar rna spontaneously undergoes relevant apo-to-holo conformational transitions in molecular dynamics and constrained geometrical simulations. *Journal of chemical information and modeling*, 50(8):1489–1501.

[59] Gonnet, P. (2007). A simple algorithm to accelerate the computation of non-bonded interactions in cell-based molecular dynamics simulations. *Journal of Computational Chemistry*, 28(2):570–573.

[64] Greengard, L. and Rokhlin, V. (1987). A fast algorithm for particle simulations. *Journal of computational physics*, 73(2):325–348.

[76] Grossfield, A. and Zuckerman, D. M. (2009). Quantifying uncertainty and sampling quality in biomolecular simulations. *Annual reports in computational chemistry*, 5:23–48.

[88] Hansen, J. S., Schrøder, T. B., and Dyre, J. C. (2012). Simplistic Coulomb Forces in Molecular Dynamics: Comparing the Wolf and Shifted-Force Approximations. *The Journal of Physical Chemistry B*, 116(19):5738–5743.

[3] Henzler-Wildman, K. and Kern, D. (2007). Dynamic personalities of proteins. *Nature*, 450(7172):964–972.

[44] Hopkins, C. W., Le Grand, S., Walker, R. C., and Roitberg, A. E. (2015). Long-Time-Step Molecular Dynamics through Hydrogen Mass Repartitioning. *Journal of Chemical Theory and Computation*, 11(4):1864–1874.

[21] Im, W. and Roux, B. (2002). Ion permeation and selectivity of ompf porin: a theoretical study based on molecular dynamics, brownian dynamics, and continuum electrodiffusion theory. *Journal of molecular biology*, 322(4):851–869.

[9] Ingólfsson, H. I., Arnarez, C., Periole, X., and Marrink, S. J. (2016). Computational 'microscopy'of cellular membranes. *J Cell Sci*, 129(2):257–268.

[14] Karplus, M. and McCammon, J. A. (2002). Molecular dynamics simulations of biomolecules. *Nature Structural & Molecular Biology*, 9(9):646–652.

[33] Kim, S. N., Kuang, Z., Slocik, J. M., Jones, S. E., Cui, Y., Farmer, B. L., McAlpine, M. C., and Naik, R. R. (2011). Preferential binding of peptides to graphene edges and planes. *Journal of the American Chemical Society*, 133(37):14480–14483.

[2] Kmiecik, S., Gront, D., Kolinski, M., Wieteska, L., Dawid, A. E., and Kolinski, A. (2016). Coarse-grained protein models and their applications. *Chem. Rev*, 116(14):7898–7936.

[67] Larsson, P., Hess, B., and Lindahl, E. (2011). Algorithm improvements for molecular dynamics simulations. *Wiley interdisciplinary reviews: computational molecular science*, 1(1):93–108.

[8] Lee, E. H., Hsin, J., Sotomayor, M., Comellas, G., and Schulten, K. (2009). Discovery through the computational microscope. *Structure*, 17(10):1295–1306.

[11] Levitt, M. (1983). Molecular dynamics of native protein. *Journal of Molecular Biology*, 168(3):595–617.

[18] Liwo, A., Khalili, M., and Scheraga, H. A. (2005). Ab initio simulations of protein-folding pathways by molecular dynamics with the united-residue model of polypeptide chains. *Proceedings of the National Academy of Sciences of the United States of America*, 102(7):2362–2367.

[75] Lyman, E. and Zuckerman, D. M. (2006). Ensemble-Based Convergence Analysis of Biomolecular Trajectories. *Biophysical Journal*, 91(1):164–172.

[78] Lyman, E. and Zuckerman, D. M. (2007). On the structural convergence of biomolecular simulations by determination of the effective sample size. *The journal of physical chemistry. B*, 111(44):12876–12882.

[25] Ma, B. and Nussinov, R. (2006). Simulations as analytical tools to understand protein aggregation and predict amyloid conformation. *Current opinion in chemical biology*, 10(5):445–452.

[49] Machado, M. R., Dans, P. D., and Pantano, S. (2011). A hybrid all-atom/coarse grain model for multiscale simulations of dna. *Physical Chemistry Chemical Physics*, 13(40):18134–18144.

[38] Malkin, V. G., Malkina, O. L., and Salahub, D. R. (1994). Calculation of spin—spin coupling constants using density functional theory. *Chemical physics letters*, 221(1-2):91–99.

[5] Matsumoto, M., Saito, S., and Ohmine, I. (2002). Molecular dynamics simulation of the ice nucleation and growth process leading to water freezing. *Nature*, 416(6879):409–413.

[23] McCammon, J. A., Gelin, B. R., and Karplus, M. (1977). Dynamics of folded proteins. *Nature*, 267(5612):585–590.

[89] McCann, B. W. and Acevedo, O. (2013). Pairwise Alternatives to Ewald Summation for Calculating Long-Range Electrostatics in Ionic Liquids. *Journal of Chemical Theory and Computation*, 9(2):944–950.

[42] Miyamoto, S. and Kollman, P. A. (1992). Settle: An analytical version of the SHAKE and RATTLE algorithm for rigid water models. *Journal of Computational Chemistry*, 13(8):952–962.

[29] Mu, Y. and Stock, G. (2006). Conformational dynamics of rna-peptide binding: a molecular dynamics simulation study. *Biophysical journal*, 90(2):391–399.

[1] O. Nielsen, S., E. Bulo, R., B. Moore, P., and Ensing, B. (2010). Recent progress in adaptive multiscale molecular dynamics simulations of soft matter. *Physical Chemistry Chemical Physics*, 12(39):12401–12414. 00000.

[66] Ohmura, I., Morimoto, G., Ohno, Y., Hasegawa, A., and Taiji, M. (2014). Mdgrape-4: a special-purpose computer system for molecular dynamics simulations. *Phil. Trans. R. Soc. A*, 372(2021):20130387.

[48] Orsi, M., Noro, M. G., and Essex, J. W. (2011). Dual-resolution molecular dynamics simulation of antimicrobials in biomembranes. *Journal of The Royal Society Interface*, 8(59):826–841.

[73] Phillips, J. C., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R. D., Kale, L., and Schulten, K. (2005). Scalable molecular dynamics with namd. *Journal of computational chemistry*, 26(16):1781–1802.

[74] Plimpton, S. (1995). Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117(1):1–19.

[50] Pluhackova, K. and Böckmann, R. A. (2015). Biomembranes in atomistic and coarse-grained simulations. *Journal of Physics: Condensed Matter*, 27(32):323103.

[52] Praprotnik, M., Delle Site, L., and Kremer, K. (2005). Adaptive resolution molecular-dynamics simulation: Changing the degrees of freedom on the fly. *The Journal of chemical physics*, 123(22):224106.

[53] Praprotnik, M., Matysiak, S., Delle Site, L., Kremer, K., and Clementi, C. (2007). Adaptive resolution simulation of liquid water. *Journal of Physics: Condensed Matter*, 19(29):292201.

[54] Praprotnik, M., Site, L. D., and Kremer, K. (2008). Multiscale simulation of soft matter: From scale bridging to adaptive resolution. *Annu. Rev. Phys. Chem.*, 59:545–571.

[32] Pumera, M., Ambrosi, A., Bonanni, A., Chng, E. L. K., and Poh, H. L. (2010). Graphene for electrochemical sensing and biosensing. *TrAC Trends in Analytical Chemistry*, 29(9):954–965.

[31] Qin, W., Li, X., Bian, W.-W., Fan, X.-J., and Qi, J.-Y. (2010). Density functional theory calculations and molecular dynamics simulations of the adsorption of biomolecules on graphene surfaces. *Biomaterials*, 31(5):1007–1016.

[17] Qiu, L., Pabit, S. A., Roitberg, A. E., and Hagen, S. J. (2002). Smaller and faster: The 20-residue trp-cage protein folds in 4 μs. *Journal of the American Chemical Society*, 124(44):12952–12953.

[80] Redon, S., Stoltz, G., and Trstanova, Z. (2016). Error Analysis of Modified Langevin Dynamics. *Journal of Statistical Physics*, pages 1–37.

[45] Riniker, S. and Gunsteren, W. F. v. (2012). Mixing coarse-grained and fine-grained water in molecular dynamics simulations of a single system. *The Journal of Chemical Physics*, 137(4):044120.

[95] Romo, T. D. and Grossfield, A. (2014). Unknown unknowns: the challenge of systematic and statistical error in molecular dynamics simulations. *Biophysical journal*, 106(8):1553–1554.

[20] Roux, B. and Karplus, M. (1994). Molecular dynamics simulations of the gramicidin channel. *Annual review of biophysics and biomolecular structure*, 23(1):731–761.

[41] Ryckaert, J.-P., Ciccotti, G., and Berendsen, H. J. C. (1977). Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23(3):327–341.

[93] Saiz, F. and Gamero-Castaño, M. (2012). Amorphization of silicon induced by nanodroplet impact: A molecular dynamics study. *Journal of Applied Physics*, 112(5):054302. 00009.

[94] Saiz, F. and Gamero-Castaño, M. (2016). Molecular dynamics of nanodroplet impact: The effect of the projectile's molecular mass on sputtering. *AIP Advances*, 6(6):065319. 00001.

[71] Salomon-Ferrer, R., Case, D. A., and Walker, R. C. (2013). An overview of the amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210.

[7] Schuh, C. A. and Lund, A. C. (2003). Atomistic basis for the plastic yield criterion of metallic glass. *Nature Materials*, 2(7):449–452.

[47] Senn, H. M. and Thiel, W. (2006). Qm/mm methods for biological systems. In *Atomistic approaches in modern biology*, pages 173–290. Springer.

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

[22] Shrivastava, I. H. and Sansom, M. S. (2000). Simulations of ion permeation through a potassium channel: molecular dynamics of kcsa in a phospholipid bilayer. *Biophysical Journal*, 78(2):557–570.

[81] Singh, K. K. and Redon, S. (2017). Adaptively restrained molecular dynamics in lammps. *Modelling and Simulation in Materials Science and Engineering*.

[16] Snow, C. D., Zagrovic, B., and Pande, V. S. (2002). The trp cage: folding kinetics and unfolded state topology via molecular dynamics simulations. *Journal of the American Chemical Society*, 124(49):14548–14549.

[51] Sokkar, P., Boulanger, E., Thiel, W., and Sanchez-Garcia, E. (2015). Hybrid quantum mechanics/molecular mechanics/coarse grained modeling: A triple-resolution approach for biomolecular systems. *Journal of chemical theory and computation*, 11(4):1809–1818.

[56] Stillinger, F. H. and Weber, T. A. (1985). Computer simulation of local order in condensed phases of silicon. *Physical review B*, 31(8):5262.

[60] Sutmann, G. and Stegailov, V. (2006). Optimization of neighbor list techniques in liquid matter simulations. *Journal of Molecular Liquids*, 125(2–3):197–203.

[57] Tersoff, J. (1988). New empirical approach for the structure and energy of covalent systems. *Physical Review B*, 37(12):6991.

[84] Trstanova, Z. and Redon, S. (2017). Estimating the speed-up of adaptively restrained langevin dynamics. *Journal of Computational Physics*, 336:412–428.

[24] Urbanc, B., Cruz, L., Ding, F., Sammond, D., Khare, S., Buldyrev, S., Stanley, H., and Dokholyan, N. (2004). Molecular dynamics simulation of amyloid β dimer formation. *Biophysical journal*, 87(4):2310–2321.

[58] Van Duin, A. C., Dasgupta, S., Lorant, F., and Goddard, W. A. (2001). Reaxff: a reactive force field for hydrocarbons. *The Journal of Physical Chemistry A*, 105(41):9396–9409.

[12] van Gunsteren, W. F. and Berendsen, H. J. C. (1990). Computer Simulation of Molecular Dynamics: Methodology, Applications, and Perspectives in Chemistry. *Angewandte Chemie International Edition in English*, 29(9):992–1023.

[85] Wolf, D., Keblinski, P., Phillpot, S. R., and Eggebrecht, J. (1999). Exact method for the simulation of Coulombic systems by spherically truncated, pairwise r-1 summation. *The Journal of Chemical Physics*, 110(17):8254–8282.

[87] Yonezawa, Y., Fukuda, I., Kamiya, N., Shimoyama, H., and Nakamura, H. (2011). Free Energy Landscapes of Alanine Dipeptide in Explicit Water Reproduced by the Force-Switching Wolf Method. *Journal of Chemical Theory and Computation*, 7(5):1484–1493. 00019.

[27] Zacharias, M. and Springer, S. (2004). Conformational flexibility of the mhc class i α 1-α 2 domain in peptide bound and free states: a molecular dynamics simulation study. *Biophysical journal*, 87(4):2203–2214.

[77] Zuckerman, D. M. (2011). Equilibrium Sampling in Biomolecular Simulation. *Annual review of biophysics*, 40:41–62.