Adaptive algorithms for molecular simulation

Svetlana Artemova

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques et Informatique**

Arrêté ministériel : 7 août 2006

Présentée par

Svetlana ARTEMOVA

Thèse dirigée par
Stéphane REDON

Préparée au sein de
NANO-D – INRIA Grenoble Rhône-Alpes
Laboratoire Jean Kuntzmann – UMR CNRS 5224

Dans l'école doctorale
Mathématiques, Sciences et Technologies de l'Information, Informatique

Algorithmes adaptatifs pour la simulation moléculaire

Thèse soutenue publiquement le 30 Mai 2012 devant le jury composé de

M. Tony LELIEVRE
Professeur a l'Ecole des Ponts ParisTech, ICPEF, Rapporteur

M. Mark TUCKERMAN
Professeur à l'Université de New York, Etats-Unis, Rapporteur

Mme. Brigitte BIDEGARAY-FESQUET
Chargée de recherche au Laboratoire Jean Kuntzmann, Examinateur

Mme. Marie-Paule CANI
Professeur à l'Institut National Polytechnique de Grenoble, Examinateur

M. Serge CROUZY
Directeur de recherche au CEA Grenoble, Examinateur

M. Richard LAVERY
Directeur de recherche à l'Institut de Biologie et Chimie des Protéines, Examinateur

M. Matej PRAPROTNIK
Maître de conférence à l'Université de Ljubljana, Slovénie, Examinateur

M. Stéphane REDON
Chargé de recherche à l'INRIA Rhône-Alpes, Directeur de thèse

Adaptive algorithms for molecular simulation

Svetlana Artemova

2012

Abstract

Molecular simulations have become an essential tool in biology, chemistry and physics. Unfortunately, they still remain computationally challenging.

In this dissertation, we propose algorithms that accelerate molecular simulations by clustering particles into rigid bodies.

We first study several neighbor-search algorithms for large rigid bodies, and show that hierarchy-based algorithms may provide significant speedups.

Accordingly, we propose a technique to build a hierarchical representation of an arbitrary molecular graph. We show how this technique can be used in adaptive torsion-angle mechanics, a simulation method that describes molecules as articulated rigid bodies.

Finally, we introduce ARPS – Adaptively Restrained Particle Simulations – a mathematically-grounded method able to switch positional degrees of freedom on and off. We propose two switching strategies, and illustrate the advantages of ARPS on various examples. In particular, we show how ARPS allow us to smoothly trade between precision and speed, and to efficiently compute correct static equilibrium properties on molecular systems.

Résume

Les simulations moléculaires sont devenues un outil essentiel en biologie, chimie et physique. Malheureusement, elles restent très coûteuses.

Dans cette thèse, nous proposons des algorithmes qui accélèrent les simulations moléculaires en regroupant des particules en plusieurs objets rigides.

Nous étudions d'abord plusieurs algorithmes de recherche de voisins dans le cas des grands objets rigides, et démontrons que les algorithmes hiérarchiques permettent d'obtenir des accélérations importantes.

En conséquence, nous proposons une technique pour construire une représentation hiérarchique d'un graphe moléculaire arbitraire. Nous démontrons l'usage de cette technique pour la mécanique adaptative en angles de torsion, une méthode de simulation qui décrit les molécules comme des objets rigides articulés.

Enfin, nous introduisons ARPS – Adaptively Restrained Particle Simulations ("Simulations de particules restreintes de façon adaptative") – une méthode mathématiquement fondée capable d'activer et de désactiver les degrés de liberté en position. Nous proposons deux stratégies d'adaptation, et illustrons les avantages de ARPS sur plusieurs exemples. En particulier, nous démontrons comment ARPS permet de choisir finement le compromis entre précision et vitesse, ainsi que de calculer rapidement des proprietés statiques d'équilibre sur les systèmes moléculaires.

Acknowledgments

The years of graduate studies have been an important experience for me, both from scientific and personal points of view, and I would like to thank everyone who made this experience positive.

First and foremost, I am deeply grateful to my supervisor Stephane Redon for accepting me as his first student, for the friendly guidance and encouragement throughout the thesis, for sharing his knowledge and ideas with me as well as for helping me to improve my writing skills. Stephane's passion for research and his commitment to the highest standards inspired me during these years.

I owe my sincere gratitude to Sergei Grudinin, for his continuous helpful cooperation and valuable advice in both theoretical and practical aspects of this work, for the critical insights that stimulated thinking and for multiple rereadings and corrections of our papers.

I have been very privileged to collaborate with Claude Le Bris, Frédéric Legoll, Tony Lelièvre, Mathias Rousset and Gabriel Stoltz during this project. They generously shared with us their experience in solving complex mathematical problems, which helped to advance the last part of the work.

I wish to thank the reviewers of this thesis and the examiners for the honor of having them in the committee, for their careful reading of the manuscript, and their useful comments and suggestions.

I must acknowledge INRIA for financing this work, hosting me and providing excellent working conditions.

I am grateful to all members of the NANO-D team (present and former), especially Evelyne Altariba, Noëlle Le Delliou, Ahmad Shahwan, Jelmer Wolterink and Jocelyn Gaté, for a friendly and cheerful environment in the group, for many enjoyable lunches and coffee breaks that we shared at INRIA, for parties and ski touring together. I warmly thank the native speakers of the team for their patience in teaching me French.

My deepest gratitude to other INRIA colleagues: Laurentiu Trifan, Lamiae Azizi, Antoine Letouzey, Michel Amat, Xavier Alameda-Pineda, Farida Enikeeva, Darren Wraith, Abdelaziz Djelouah, Diana Stefan for many pleasant moments that we spent together.

I also greatly appreciate the help of Françoise De Coninck with administrative problems.

I would like to single out Maël Bosson for being an amusing office-mate and a great friend, for helping me to meet a lot of interesting people in Grenoble and introducing me to various mountain activities.

I would certainly like to thank Sebastien Letout, my housemate, for helping me discover a completely different way of thinking and living, Vasil Khalidov and Olga Leshchishina for being caring and dynamic friends always ready to propose something interesting to do or something delicious to eat at their place, Caroline Richard for many joyful evenings and mountain hikes together, for her inexhaustible energy and wise support, as well as Olga Nagornaya, Olivier Tosoni, Laure Malagnoux, Dmitriy Gritskevich, Liaissan Fatikhova, Florence Gabarrou, Simon Courtemanche, Franz Schuster, Tanya Gordeliy, Marina Postnikova, Karine Kuyumzhiyan and many others for making my life in France more colorful.

I warmly thank my friends in Russia: Anna Sinelshchikova, Natasha Kalyakina, Marina and Roman Igla, Irina and Vladimir Khoroshiltsev, Ivan Ivanov, Anna and Artem Gordeev, Dmitriy Artemov, Oxana Kramareva for still staying close and always heartily hosting me when I am back home.

Of course, I am grateful to my parents, Galina and Mikhail, for giving me the opportunity to do all the studies that made me qualified to enter this PhD program, for teaching me not to be afraid of difficulties and always encouraging me, to my sister Tanya, my closest friend and an example to follow, and to my family-in-law (Christine, Jean-Marc, Constance, Edouard, Claire) for their moral support and presence.

Finally, I am indebted to François-Xavier Boillot, my dear husband, for bringing joy and serenity to my life, for his invaluable patience (especially during the last months of the thesis), and love and support.

A la mémoire de Mica Boirat,
ami, coloc et grimpeur d'exception.

Contents

Abstract iii

Acknowledgments v

Table of Contents xiii

List of Tables xv

List of Figures xxv

List of Algorithms xxvii

Related publications xxix

1 Introduction (Français) 1

1.1 Représentation d'un système moléculaire . . . 2

1.1.1 Dynamique des particules . . . 2

1.1.2 Dynamique des corps rigides et articulés . . . 2

1.2 Simulation de systèmes moléculaires . . . 3

1.2.1 L'énergie potentielle et les forces . . . 4

1.2.2 Génération de configurations . . . 5

1.2.2.1 Dynamique moléculaire . . . 5

1.2.2.2 Monte Carlo . . . 5

1.3 Simulations efficaces . . . 5

1.3.1 Choix d'un modèle approprié . . . 5

1.3.1.1 Représentation tout-atome . . . 6

1.3.1.2 Représentation des atomes unifiés . . . 6

1.3.1.3 Représentation atomistique, réduction du nombre de degrés de liberté . . . 6

1.3.1.4 Représentation gros-grain et DPD . . . 7

1.3.1.5 Représentations hybrides . . . 7

1.3.2 Calculs d'énergie et de forces . . . 8

1.3.2.1 Approximation des forces . . . 8
1.3.2.2 Mise à jour partielle . . . 8
1.3.3 Génération de configurations . . . 9
1.3.3.1 Dynamique moléculaire et intégration efficace . . . 9
1.3.3.2 Monte-Carlo et échantillonnage efficace . . . 10
1.3.4 Matériel informatique . . . 11
1.3.5 Logiciels . . . 11
1.4 Contributions . . . 12

1 Introduction **15**
1.1 Representing a molecular system . . . 15
1.1.1 Particle dynamics . . . 16
1.1.2 Rigid-body/Articulated-body dynamics . . . 16
1.2 Simulating molecular systems . . . 17
1.2.1 Potential energy and forces . . . 18
1.2.2 Generating configurations . . . 18
1.2.2.1 Molecular dynamics . . . 18
1.2.2.2 Monte Carlo . . . 19
1.3 Efficient simulations . . . 19
1.3.1 Choosing an appropriate model . . . 19
1.3.1.1 All-atom representation . . . 19
1.3.1.2 United-atom representation . . . 20
1.3.1.3 Atomistic representation, reduced number of degrees of freedom . . . 20
1.3.1.4 Coarse-graining and DPD . . . 21
1.3.1.5 Mixing representations . . . 21
1.3.2 Computing energies and forces . . . 22
1.3.2.1 Approximating forces . . . 22
1.3.2.2 Partial update . . . 22
1.3.3 Generating configurations . . . 22
1.3.3.1 Molecular dynamics and efficient integration . . . 23
1.3.3.2 Monte Carlo and efficient sampling . . . 24
1.3.4 Hardware . . . 24
1.3.5 Software . . . 25
1.4 Contributions . . . 25

2 Neighbor search algorithms for large rigid molecules **29**
2.1 Introduction . . . 31
2.2 Neighbor-search algorithms . . . 32
2.2.1 Grid-based algorithm . . . 32
2.2.2 Hierarchy-based algorithms . . . 33
2.2.2.1 Construction of the binary hierarchy . . . 33
2.2.2.2 Neighbor search algorithm . . . 36

2.2.2.3 Proximity queries for bounding volumes . . . 38
2.3 Results . . . 41
2.3.1 Basic operation time . . . 42
2.3.2 Preprocessing time . . . 44
2.3.3 Cutoff test . . . 46
2.3.3.1 Choice of leaf size parameter m . . . 46
2.3.3.2 Biological complexes . . . 46
2.3.3.3 Two bluetongue virus capsids . . . 48
2.3.3.4 Two apoferritin molecules . . . 48
2.3.4 Distance test . . . 49
2.3.4.1 Biological complexes . . . 50
2.3.4.2 Two bluetongue virus capsids . . . 50
2.3.4.3 Two apoferritin molecules . . . 51
2.3.5 Remarks . . . 52
2.4 Conclusion . . . 54

3 Fast construction of assembly trees for molecular graphs 57
3.1 Introduction . . . 59
3.2 Basic definitions and problem statement . . . 60
3.3 Algorithm description . . . 61
3.3.1 Contracting cycles . . . 61
3.3.2 Building assembly trees for connected components . . . 64
3.3.3 Building the assembly tree of the complete graph . . . 65
3.4 Algorithm analysis . . . 68
3.4.1 Step 1: cycle contraction . . . 69
3.4.2 Step 2: building an assembly tree for a connected component . . . 69
3.4.3 Step 3: building an assembly tree for the whole system . . . 70
3.5 Implementation and results . . . 71
3.5.1 Difficult cases . . . 71
3.5.1.1 Step 1 . . . 72
3.5.1.2 Step 2 . . . 72
3.5.1.3 Step 3 . . . 73
3.5.2 Molecular graphs . . . 75
3.5.2.1 Step 1 . . . 78
3.5.2.2 Step 2 . . . 78
3.5.2.3 Step 3 . . . 78
3.6 Applications of the algorithm . . . 80
3.6.1 Adaptive torsion-angle molecular quasi-statics . . . 80
3.6.2 Other applications . . . 86
3.7 Conclusion . . . 86

4 ARPS: Adaptively Restrained Particle Simulations **87**
4.1 Introduction . . . 89
4.2 Theory . . . 90
4.2.1 General Hamiltonian . . . 90
4.2.2 Statistics . . . 91
4.2.3 Integration . . . 93
4.2.3.1 Integration in the NVE ensemble . . . 93
4.2.3.2 Integration in the NVT ensemble . . . 95
4.2.4 Choosing the inverse mass matrix . . . 95
4.2.4.1 How to switch . . . 95
4.2.4.2 When to switch . . . 96
4.3 Discussion . . . 97
4.3.1 Phase portrait . . . 97
4.3.2 Rescaling time . . . 98
4.3.3 Temperature in the NVT ensemble . . . 102
4.4 Algorithms . . . 104
4.4.1 Force update, first method . . . 106
4.4.2 Force update, second method . . . 109
4.5 Results . . . 112
4.5.1 Argon liquid . . . 112
4.5.2 Collision cascade . . . 114
4.5.2.1 Collision cascade in 1D . . . 117
4.5.2.2 Collision cascade in 2D . . . 119
4.5.2.3 Collision cascade in 3D . . . 121
4.5.3 Radial distribution function . . . 123
4.5.4 Polymer in solvent . . . 126
4.6 Conclusion . . . 130

5 Hierarchical ARPS **131**
5.1 Introduction . . . 133
5.2 Theory . . . 134
5.2.1 Choice of inverse mass matrix . . . 134
5.2.1.1 How to switch . . . 134
5.2.1.2 When to switch . . . 135
5.3 Discussion . . . 137
5.4 Algorithms . . . 139
5.4.1 Updating forces . . . 139
5.4.2 Updating momenta . . . 141
5.4.3 Updating positions . . . 141
5.5 Results . . . 145
5.5.1 Collision cascade in 2D . . . 145
5.5.2 Radial distribution function . . . 146

5.5.3 Discussion on stability 147
5.6 Conclusion 147

6 Conclusions and outlook **149**

6 Conclusions et perspectives (Français) **153**

Bibliography **157**

List of Tables

2.1 Comparing neighbor search algorithms for large rigid molecules. Summary of the information on the benchmark systems. 42
2.2 Timings necessary for hierarchy-based algorithms to perform a basic operation: compare a distance between two bounding volumes with a cutoff value. 43
2.3 Experimentally obtained values for the average ratio r for different values of the cutoff and two types of bounding volumes. Corresponding values of parameter m. 56

4.1 Summary of the information on four simulations of a periodic 3D box of 21 952 Argon particles, with different pairs of thresholds for AR simulations (NVE ensemble). Average number of forces updated each time step $\langle N_F \rangle$, average number of active particles $\langle N_{act} \rangle$, average wall-clock time per time steps $\langle t \rangle$ may be significantly reduced. 117
4.2 Summary of the information on three simulations of the collision cascade in 1D, NVE ensemble. Large shock. 119
4.3 Summary of the information on three simulations of the collision cascade in 1D, NVE ensemble. Small shock. 119
4.4 Average values of the polymer parameters over reference and AR simulations, relative errors for these parameters, and the speedups. For each parameter the speedup was computed, as described in the text, from the coefficients a of the fitting functions of the form $a/\sqrt{x+b}$. 128

List of Figures

1.3.1 Molecular simulations on different time-length scales. Picture taken from the Lecture Notes to MolSim 2012. 26

2.2.1 Grid-based algorithm for neighbor list construction in 2D. **Top figure**: method overview. The receptor molecule is green (left molecule), the ligand molecule is brown (right molecule). **Bottom figure**: the three main steps of the grid-based neighbor search algorithm. **A**. Method initialization: receptor atoms (green or light grey circles) are distributed to the grid. **B**. Each atom of the ligand molecule (brown or dark grey circle) is individually mapped to the grid. Receptor atoms interacting with this ligand atom may only be situated in the cells adjacent to the cell of the ligand atom (these cells to be examined are enclosed into the red square). Receptor atoms interacting with the ligand atom are situated within the cutoff distance circle (red circle). Pairs of atoms to be included into the neighbor list are marked with grey dashed lines. **C**. Result of the algorithm's execution: all ligand atoms are mapped to the grid. All possible pairs to be included into the neighbor list are marked with grey dashed lines. 34

2.2.2 Hierarchical representation of a molecule. A. The whole molecule and the bounding volume enclosing it correspond to the root node of the tree. B. The two halves of the initial molecule and the two boxes enclosing these halves correspond to the child nodes of the root node. C. Halves of the halves of the initial molecule and their enclosing boxes correspond to internal nodes of the tree. For visual clarity bounding boxes are not tight. 35

2.2.3 Two levels of the binary BV-hierarchy in 2D: for AABB's (left) and OBB's (right). BV's of a higher level of the hierarchy are shown with solid lines, child BV's of a lower level are represented by dashed lines. 37

2.2.4 Two levels of the binary spherical BV-hierarchy in 2D: two child spheres are bounded (layered hierarchy) or the tightest bounding circle for the whole system is computed (wrapped hierarchy). BV's of a higher level of the hierarchy are shown with solid lines, child BV's of a lower level are represented by dashed lines. 37

2.2.5 Neighbor search for two rigid bodies using a hierarchy of oriented bounding volumes (zero cutoff). Algorithm stops when bounding volumes corresponding to different rigid bodies do not overlap (level 4 on the picture), otherwise it goes down the hierarchy. 40

2.3.1 Ribonuclease Sa complex with barstar (PDB code 1AY7, chain A is green, chain B is yellow) and 70s ribosome (PDB codes 3MS1, yellow, and 3MR8, green). Images obtained with PyMol [1]. 41

2.3.2 Top: Bluetongue virus capsid asymmetric unit (PDB code 2BTV) and the complete virus capsid. Bottom: apoferritin molecule (asymmetric unit code PDB code 1AEW). Images obtained with PyMol [1] and VMD [2]. 43

2.3.3 Left plot: preprocessing time of the grid algorithm as a function of the cutoff distance (left axis), and the number of the cells in the constructed grid as a function of the cutoff distance (right axis). Right plot: average number of atoms in occupied grid cells, solid line represents a cubic fit. 45

2.3.4 Preprocessing times of all hierarchy-based algorithms as functions of m (left axis). Corresponding true average number of atoms in the leaf BV of the hierarchy (right axis). 45

2.3.5 Cutoff test. Time dependence of AABB- and OBB-based algorithms (left) and the two algorithms based on spheres (right) on the leaf size parameter m. Cutoff distances expressed in Angstroms. 47

2.3.6 Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact for Sa-barstar complex (left) and 70s ribosome (right). Solid lines represent fitting curves. 48

2.3.7 Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact for two bluetongue virus capsids in logarithmic scale (left), and of only hierarchy-based algorithms in linear scale (right). Solid lines represent fitting curves. 49

2.3.8 Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact without rotations (left) and with rotations (right) of one of the two apoferritin molecules. Solid lines represent fitting curves, except for the one corresponding to the grid algorithm timings on the left plot. Error bars on the right plot represent maximal and minimal values. 50

2.3.9 Distance test. Computational performances of all algorithms as functions of the number of pairs in contact within the fixed cutoff for Sa-barstar complex (left) and 70s ribosome (right). Solid lines represent fitting curves, except for the one corresponding to the grid algorithm timings on the right plot. . . 51

2.3.10 Distance test. Computational performances of all algorithms as functions of the number of pairs in contact for two bluetongue virus capsids in logarithmic scale (left), and for only hierarchy-based algorithms in linear scale (right). Solid lines represent fitting curves. 52

2.3.11 Distance test. Computational performances of all algorithms as functions of the number of pairs in contact without rotations of one of two apoferritin molecules (left) and with rotations (right). Solid lines represent fitting curves. Error bars on the right plot represent maximal and minimal values. 53

2.3.12 Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact for the system of two apoferritin molecules. Results obtained on Computer 1 (left) and on Computer 2 (right). Solid lines represent fitting curves, except for those for the grid algorithm. 53

3.1.1 Example of a molecular system (left) and a corresponding molecular graph (right). Graph vertices are atoms, graph edges are covalent bonds, graph connected components are molecules. 59

3.1.2 Assembly tree corresponding to a graph. Blue nodes represent vertices of the input graph and the tree nodes corresponding to them (they have indices from 1 to 3). Yellow nodes describe internal nodes of the tree. The red node is the root node of the tree. Internal nodes and the root node are marked according to the content of the corresponding sub-graph: for example, the node marked "1+2" corresponds to a sub-graph containing vertices 1 and 2. Dashed grey arrows indicate sub-graphs corresponding to the internal node of the tree and the root node. 60

3.3.1 Workflow of the first and the second steps of the tree construction algorithm, overview. Blue nodes represent vertices of the input graph (input for steps 1 and 2), and tree nodes, corresponding to them (output of step 2); white nodes stand for group vertices (output of step 1) and tree nodes, corresponding to them (output of step 2); yellow nodes denote internal nodes of the final tree, the red node corresponds to the root node of the tree. Dashed lines indicate the cycles to be contracted. Relationship between parent and child nodes in a tree is indicated by arrows. 62

3.3.2 The workflow of the complete tree construction algorithm for the general molecular graph containing several connected components. Blue nodes represent vertices of the input graph (input for steps 1 and 2), and tree nodes, corresponding to them (output of step 2); white nodes stand for group vertices (output of step 1) and tree nodes, corresponding to them (output of step 2); yellow nodes denote internal nodes of the final tree; red nodes represent root nodes of the trees corresponding to connected components of the graph (output of step 2 and input for step 3). Relationship between parent and child nodes in a tree is indicated by arrows. 63

3.3.3 Workflow of the first step of the tree construction algorithm, example of a cycle contraction. 64

3.3.4 The second step of the tree construction algorithm, construction of an unnecessarily unbalanced tree using non-modified algorithm. Blue nodes represent vertices of the input graph and tree nodes, corresponding to them; white nodes stand for vertices representing contracted sub-graphs and correspond to the internal nodes of the tree; yellow nodes denote internal nodes of the final tree; the red node represents the root node of the tree. Dashed edges are the edges to be contracted during the next pass. 65

3.3.5 The second step of the tree construction algorithm, a big branching factor example: the tree is always unbalanced. Blue nodes represent vertices of the input graph, and tree nodes, corresponding to them; yellow nodes describe internal nodes of the final tree; the red node represents the root node of the tree. 66

3.3.6 The second step of the tree construction algorithm, construction of an assembly tree using modified algorithm. Blue nodes represent vertices of the input graph and tree nodes, corresponding to them; white nodes stand for vertices representing contracted sub-graphs and correspond to the internal nodes of the tree; yellow nodes denote internal nodes of the final tree; the red node represents the root node of the tree. Dashed edges are the edges to be contracted during the next pass. 66

3.3.7 Hierarchy of cells in 2D. Parent-child relationship between cells. Detailed description in the text. 67

3.3.8 The third step of the tree construction algorithm, algorithm iteration for a non-empty cell in 2D. Red nodes represent root nodes of the sub-graphs corresponding to the connected components of the input graph; yellow nodes describe internal nodes of the final tree. 68

3.4.1 The third step of the tree construction algorithm, planar example: the chosen number of cells is too big. Red nodes represent root nodes of the sub-graphs corresponding to the connected components of the input graph. Different types of lines (bold, dashed) correspond to different levels of the hierarchy of cells. 71

3.5.1 Computational performance of the algorithm's first step on finite cubical 3D lattices. Running time as a function of the number of edges in the input graph is plotted. The solid line represents a linear fit. 72

3.5.2 Computational performance of the algorithm's second step on unbranched chains. Running time as a function of the number of edges in the graph is plotted. The solid line represents a linear fit. 73

3.5.3 Running time dependence on the graph branching factor for the second step of the new algorithm. Random graphs with different number of edges are considered and correspond to solid lines (inverse law fits). 74

3.5.4 Average depth of the assembly tree leaf nodes for the second step of the new algorithm, plotted as a function of number of edges in the input graph (random graphs with a fixed branching factor were used). The black line corresponds to our algorithm, the grey line to recursive splitting in halves. . 74

3.5.5 Computational performance of the algorithm's third step on uniformly distributed systems in 3D. Running time as a function of the number of connected components in the input graph is plotted. The solid line represents a linear fit. 75

3.5.6 The third step of the algorithm, comparison with $O(N \log N)$ algorithms on uniformly distributed systems in 3D. Running time for all three algorithms is plotted as a function of number of connected components in the input graph. Lines on the figures are fitting curves, circles stand for our algorithm, triangles for the $O(N \log N)$ algorithm with the nth-element used for splitting, squares for the $O(N \log N)$ algorithm with the median of medians used for splitting. On the inset results are presented in logarithmic scale: the firm line stands for our algorithm, the dotted line for the $O(N \log N)$ algorithm with the nth-element used for splitting, the dashed line for the $O(N \log N)$ algorithm with the median of medians used for splitting. 76

3.5.7 The third step of the algorithm, comparison with $O(N \log N)$ algorithms on uniformly distributed systems in 3D. Average depth of the assembly tree leaf nodes is plotted as a function of number of connected components in the input graph. Circles stand for our algorithm, squares for both $O(N \log N)$ algorithms (same results for the two versions of splitting techniques). 76

3.5.8 PDB file, simple example. 77

3.5.9 Running time for passes one and two (left and right figures respectively) of the algorithm's first step on the subset of PDB molecular graphs is plotted as a function of the number of edges in the input graph. Solid lines represent linear fits, error bars indicate the standard deviation for timings in each interval. 79

3.5.10 Running time of step two of the algorithm on the subset of PDB molecular graphs is plotted as a function of the number of edges in the input graph. The solid line represents a linear fit, error bars indicate the standard deviation for timings in each interval. 79

3.5.11 Running time of the algorithm's step three on the full set of PDB molecular graphs is plotted as a function of the number of edges in the input graph. Solid lines represent linear fits, error bars indicate minimum and maximum times in each interval. 80

3.5.12 Examples of assembly trees constructed for molecular graphs: a) the assembly tree corresponding to a single connected component, a helix; b) the assembly tree corresponding to a water box. 81

3.6.1 Assembly tree of a short polypeptide. The left part (a) shows the assembly tree of a tetra-alanine (CHARMM19 model). Colors indicate the correspondence between leaf nodes and rigid bodies. The right part (b) shows a possible active region (green nodes) produced by the adaptive algorithm during the simulation, when the user has allowed for three simultaneously active joints only. This results in a partitioning of the molecule into four rigid bodies (one color per rigid or rigidified body — colors also indicate the correspondence between rigid bodies and leaf or internal nodes). 84

3.6.2 Adaptivity example, the force is applied to a poly-alanine (left) and to a part of a protein complex (PDB code 1AC1, right). Colors represent rigid body clusters (hence show the location of active torsion angles). For the right molecule the point where the force is applied is marked with a red circle. . . 85

3.6.3 Interactive docking example. Degrees of freedom are represented by different colors. 85

4.2.1 Restraining function ρ as a function of particle's kinetic energy $K(p)$. 96

4.3.1 Phase portrait of the harmonic oscillator in 1D, full-dynamics simulation. . . 98

4.3.2 Phase portrait of the adaptively restrained harmonic oscillator in 1D ($\varepsilon_1^r =$ 0.5 kcal/mol, $\varepsilon_1^f = 0.8$ kcal/mol). Opal lines stand for isolines of this Hamiltonian, the thick black line for a specific isoline of this Hamiltonian ($H_{AR} \equiv 1$), the dotted red circle represents the corresponding isovalue of the classical Hamiltonian ($H \equiv 1$). In the full-dynamics region, trajectories remain unperturbed. In the restrained-dynamics region, positions are constant although momenta evolve. In the transition region trajectories smoothly switch between restrained dynamics and full dynamics. 99

4.3.3 Phase portraits of the adaptively restrained harmonic oscillator for several sets of thresholds, ε stands for ε_1^r (in kcal/mol), the width of the transition region δ expressed in kcal/mol. 100

4.3.4 Projection of the AR trajectory to the full-dynamics trajectory. Detailed description is provided in the text. 101

4.3.5 Period recovering for 1D harmonic oscillator. Top picture parameters: $\varepsilon^f =$ 0.002 kcal/mol, $\delta = 0.0015$ kcal/mol. Bottom picture parameters: $\varepsilon^f = 0.005$ kcal/mol, $\delta = 0.0025$ kcal/mol. Initial momentum of the particle was 0.089. . 103

4.3.6 Dependence of the temperature T^* (eq. 4.3.1) (in AR simulation $\varepsilon^r = 1$ kcal/mol, $\varepsilon^f = 2$ kcal/mol) on the temperature of the thermostat T for a system of 1 particle ($m = 1$) in 3D. 104

4.3.7 System of 1 particle ($m = 1$) in 3D. Dependence of the temperature T^* (eq. 4.3.1) on the thresholds ε^r and ε^f for a fixed thermostat temperature $T = 95$ K. 105

4.4.1 ARPS. Force update, second method. **Top figure**: an input for the force update algorithm is the system configuration from the previous time step. The interactions between the particles (interaction forces between pairs of atoms) at the previous time step are displayed by arrows. The particles though, are colored according to the restraining function calculated at the *current* time step: active particles are green (light grey), fully-restrained particles are blue (dark grey). **Bottom figure**: the three main steps of the algorithm. **1.** The first step of the algorithm subtracts interactions from the two particles in each pair where at least one particle is active (the corresponding arrows disappear). **2.** The second step of the algorithm updates the grid according to the positions of active particles at the current time step (only these particles have moved). **3.** The third step of the algorithm adds interactions due to the new positions of the active particles to both particles in each pair where at least one particle is active (new arrows corresponding to the new forces appear). 110

4.4.2 Possible modification of a 2D-grid in the second force-update method to introduce periodic boundary conditions. The green dashed line surrounds the grid cells that may be occupied by the system's particles, and the cells outside this line (buffer cells) are always empty. Highlighted cells of the same color (red and blue) represent mutual mirror cells. If during the algorithm's execution, while examining neighboring cells of some cell (*e.g.* cell 1 on the figure), a buffer cell is found (cell 2), its mirror cell (cell 3) will be considered instead. 112

4.5.1 Lennard-Jones potential, normal $V_{LJ}(r)$ (firm line) and truncated between 1.5 and 2 Å $V_{LJ}^{tr}(r)$ (dashed line). 113

4.5.2 Four simulations of a periodic 3D box of 21 952 Argon particles (NVE ensemble): reference simulation and 3 AR simulations with different pairs of thresholds (in kcal/mol). We output total full-dynamics energy (in kcal/mol) for the reference simulation and total adaptive energies for AR simulations. . 114

4.5.3 Four simulations of a periodic 3D box of 21 952 Argon particles, with different pairs of thresholds (in kcal/mol) for AR simulations (NVE ensemble): number of forces updated each time step (top) and number of active particles at each time step (down). 115

4.5.4 Simulations of a periodic 3D box of 21 952 Argon particles (NVE ensemble): linear momenta evolution for the full-dynamics simulation (top) and one AR simulation (down). 116

4.5.5 Simulating a collision cascade in the 1D system of 100 particles in the NVE ensemble (thresholds in kcal/mol). Shock front for a large shock (top) and a small shock (down). 118

4.5.6 Simulating a collision cascade in 2D in the NVE ensemble with controlled precision (thresholds in kcal/mol). Adaptively restrained simulations allow us to smoothly trade between precision and speed. Even for large speedups (up to 10x) the features of the shock are extremely well preserved. 120

4.5.7 Simulating a collision cascade in a 2D system in the NVE ensemble, error analysis: particle displacements A_i (top), particle displacements relative to reference (middle); particle displacements R_i relative to the displacements in the reference simulation(bottom). Thresholds are expressed in kcal/mol. Error appears as in the zones were the particles move the most, as on the border of the shock wave. 122

4.5.8 Collision cascade in a 3D system of 138 916 particles in the NVE ensemble. For visual clarity only a half of the particles is displayed. Adaptively restrained simulations allow us to smoothly trade between precision and speed. Thresholds for AR simulations from left to right (in kcal/mol): $\varepsilon^r = 0$, $\varepsilon^f = 0.01$ (speedup 7.5); $\varepsilon^r = 0.01$, $\varepsilon^f = 0.02$ (speedup 10); $\varepsilon^r = 99$, $\varepsilon^f = 100$ (speedup 25). 123

4.5.9 Radial distribution functions for an Argon system (periodic 3D box, 343 particles, NVT ensemble). Even though the AR simulation has significantly modified the system's dynamics (see $\langle N_{act} \rangle$), the curves coincide. 125

4.5.10 Radial distribution functions of an Argon system obtained with full-dynamics and AR simulations (periodic 3D box, 343 particles, NVT ensemble). After 50 000 simulation steps RDF in the AR simulation is noisier than the reference one. 125

4.5.11 Computing the hydrodynamic radius R_H of a solvated polymer, NVT ensemble. Full-dynamics simulations reduce the variance more at each time step (top), but AR simulations perform many more time steps, so that they reduce the variance faster in wall-clock time (bottom). For any target precision, AR simulations compute the hydrodynamic radius four times faster than full-dynamics simulations. 127

4.5.12 Active density of the solvent for the cases of 2,3 and 5 monomers in a polymer (NVT ensemble). The solid line stands for 2 monomers, the dashed line for 3 monomers, the dotted line for 5 monomers. 129

5.3.1 Trajectories in space of the two particles connected by a stiff spring in 1D during a full-dynamics simulation. 138

5.3.2 Trajectories of two particles connected by a stiff spring in 1D during hierarchical AR simulations with different sets of thresholds. 138

5.5.1 Hierarchical ARPS. Simulating a collision cascade with controlled precision. Hierarchical adaptively restrained simulations allow us to smoothly trade between precision and speed. The main features of the shock are preserved. The binary tree representation was constructed top-down. 145

5.5.2 Hierarchical ARPS. Simulating a collision cascade with controlled precision. Hierarchical adaptively restrained simulations allow us to smoothly trade between precision and speed. The main features of the shock are preserved. The binary tree representation was constructed bottom-up. 146
5.5.3 Hierarchical ARPS. Radial distribution functions for an Argon system (periodic 3D box, 343 particles, NVT ensemble). The curves corresponding to the reference and the AR simulations coincide. 147

List of Algorithms

2.1 Algorithm for computing a neighbor list for two rigid bodies with a certain cutoff distance using a grid. . . . 33
2.2 Algorithm for computing a neighbor list for two rigid bodies within a certain cutoff distance using a corresponding hierarchy of axis-aligned, oriented or spherical bounding volumes. . . . 39
3.1 Complete description of the algorithm for constructing an assembly tree from a molecular graph in pseudo-code. . . . 69
4.1 ARPS, NVE ensemble, integration scheme. Lower indices stand for time step numbers, the time step size is denoted by h. . . . 106
4.2 ARPS, the first method for incremental force update. Lower indices stand for time step numbers, upper indices are particle indices, a denotes a particle, q denotes its position, f its force, r^{ij} is a radius-vector connecting particles a^i and a^j, and the function *calculateForce()* is specific for each potential. . . . 108
4.3 ARPS, the second method for the incremental force update. In this Algorithm a denotes a particle, q denotes its position, f its force, r_n^{ij} is a radius-vector connecting particles a^i and a^j at time step n, and the function *calculateForce()* is specific for each potential. . . . 111
5.1 Hierarchical ARPS. General integration scheme for simulations in the NVE ensemble. . . . 139
5.2 Hierarchical ARPS. Main steps of the force-update algorithm. . . . 141
5.3 Hierarchical ARPS. Algorithm for the particle momenta update. Lower indices stand for time step numbers, the time step size is denoted by h. . . . 141
5.4 Hierarchical ARPS. Updating structures $\mathbf{Q}_C$, $\mathbf{R}_C$ and $\mathbf{S}_C$. Each internal node C is considered to have two child nodes A and B. . . . 144
5.5 Hierarchical ARPS. Updating positions. Each internal node C is considered to have two child nodes A and B. Lower indices stand for time step, upper indices are particles indices, q denotes particles positions, h denotes the time step size. 144

Related publications

1. **Artemova, S.**, Grudinin, S. and Redon, S. (2011), Fast construction of assembly trees for molecular graphs. Journal of Computational Chemistry, 32: 1589–1598. doi: 10.1002/jcc.21738. Full text.

2. **Artemova, S.**, Grudinin, S. and Redon, S. (2011), A comparison of neighbor search algorithms for large rigid molecules. Journal of Computational Chemistry, 32: 2865–2877. doi: 10.1002/jcc.21868. Full text.

3. **Artemova, S.**, Redon S., (2012), ARPS: Adaptively restrained particle simulations. Physical Review Letters. doi: 10.1103/PhysRevLett.109.190201. Full text.

4. Patent for ARPS, deposited.

5. Patent for hierarchical ARPS, deposited.

Chapter 1

Introduction (Français)

Les simulations moléculaires jouent un rôle de plus en plus important dans la recherche sur la matière à l'échelle nanométrique. Elles permettent de comprendre de nombreux processus chimiques et biologiques dans différents matériaux [3, 4] et dans les cellules vivantes [5, 6], d'observer de nouveaux phénomènes, de découvrir les caractéristiques de la matière physique [7], ainsi que de prédire le comportement des systèmes à l'étude [8]. Les simulations moléculaires peuvent également aider à tester de nouveaux modèles physiques et à expliquer des phénomènes encore mal connus. Elles permettent la modélisation et la conception de molécules aux propriétés voulues [9], la simulation sous des conditions et des états difficiles à créer en laboratoire [10] tout en ajustant facilement les paramètres du système. En conséquence, les simulations moléculaires sont aujourd'hui largement utilisées dans de nombreux domaines de recherche, par exemple la chimie, la physique et la biologie numériques, ainsi que la science des matériaux. Les résultats obtenus peuvent avoir des applications directes dans des domaines majeurs tels que la conception de médicaments [11, 12, 13, 14], les nanotechnologies [15], l'électronique [16], l'optique, etc.

Malgré cela, la simulation de nombreux problèmes importants de la chimie, la biologie et la physique, comme par exemple le repliement de protéines [17, 18], la solvatation de molécules [19], la diffusion à travers les membranes biologiques [20], les réactions chimiques [19], etc. demeure un défi. Cette thèse décrit plusieurs algorithmes visant à accélérer la simulation moléculaire pour résoudre ces problèmes.

Les simulations moléculaires consistent en la création d'une représentation (modèle) du système moléculaire sur l'ordinateur et, ensuite, en la génération d'un certain nombre de configurations réalistes de ce système. A partir de ces configurations, différents types d'informations peuvent être extraites : les propriétés thermodynamiques, les propriétés dynamiques, les particularités structurelles, etc.

Nous présentons dans un premier temps plusieurs représentations possibles des systèmes moléculaires.

1.1 Représentation d'un système moléculaire

Il existe plusieurs façons de représenter un système moléculaire. La mécanique quantique, par exemple, considère ce système comme une collection d'atomes et de particules subatomiques, et décrit leurs propriétés par une fonction d'onde. À plus grande échelle, des atomes, des groupes d'atomes ou des molécules peuvent être représentés comme des objets distincts qui interagissent à travers une fonction (empirique) de potentiel. Enfin, un système pourrait être modélisé comme une masse continue : fluide ou solide.

Dans cette thèse, nous nous intéressons à la représentation discrète des systèmes moléculaires dans le cadre de la mécanique classique, plus précisément les particules, les corps rigides et les corps articulés, et nous envisageons leur dynamique.

1.1.1 Dynamique des particules

Un système moléculaire en 3D peut être considéré comme une collection de N particules (des atomes par exemple) représentées par des masses ponctuelles. L'utilisation d'une fonction hamiltonienne $H(\mathbf{q}, \mathbf{p})$, où $\mathbf{q}$ et $\mathbf{p}$ sont les vecteurs de dimension $3N$ de positions et moments des particules respectivement, est une des approches possibles pour représenter un tel système. Un hamiltonien est généralement exprimé comme la somme des énergies potentielle et cinétique du système : $H = K + V$. La forme précise de ces énergies est choisie selon le système à l'étude. Une forme répandue du hamiltonien est:

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

où $\mathbf{M}$ est une matrice de masse diagonale de taille $3N \times 3N$ (qui peut dépendre de positions $\mathbf{q}$), et $V(\mathbf{q})$ est un potentiel d'interaction.

Les équations du mouvement peuvent être déduites de ce hamiltonien:

$$\begin{aligned}\dot{\mathbf{p}} &= -\frac{\partial H(\mathbf{q}, \mathbf{p})}{\partial \mathbf{q}} = -\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}}, \\ \dot{\mathbf{q}} &= \frac{\partial H(\mathbf{q}, \mathbf{p})}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p}.\end{aligned}$$

1.1.2 Dynamique des corps rigides et articulés

Les corps rigides, contrairement aux particules, occupent un certain volume dans l'espace et possèdent des propriétés géométriques comme, par exemple, un centre de masse ou des moments d'inertie. Les corps rigides sont considérés comme non déformables. Alors qu'en 3D, les particules ont 3 degrés de liberté en translation, les corps rigides, eux, en ont 6 (translation et rotation).

Pour écrire les équations du mouvement d'un corps rigide, il est commode d'utiliser les notations spatiales [21]. Ces notations décrivent la vitesse, l'accélération, l'inertie des corps

rigides en utilisant des vecteurs et tenseurs en 6D. L'équation du mouvement d'un corps rigide peut alors être écrite comme suit:

$$\mathbf{f} = \frac{d}{dt}(\mathbf{Iv}) = \mathbf{Ia} + \mathbf{v} \times \mathbf{Iv},$$

où $\mathbf{f}$ est la force spatiale agissant sur un corps rigide, $\mathbf{I}$ est l'inertie spatiale du corps rigide, $\mathbf{v}$, $\mathbf{Iv}$, et $\mathbf{a}$ sont sa vitesse spatiale, son moment spatial et son accélération spatiale respectivement, et "$\times$" représente un produit vectoriel spatial. Par conséquent, l'accélération d'un corps rigide peut être écrite:

$$\mathbf{a} = \mathbf{I}^{-1}\mathbf{f} - \mathbf{I}^{-1}(\mathbf{v} \times \mathbf{Iv}) = \Phi\mathbf{f} + \mathbf{b},$$

où Φ est l'inertie spatiale inverse du corps rigide, et $\mathbf{b}$ est le biais d'accélération (c'est à dire l'accélération du corps rigide lorsqu'aucune force ne lui est appliquée).

Les corps articulés, largement utilisés aussi en robotique et en infographie, sont des corps rigides avec contraintes. Il a été démontré [21, 22] que la dynamique d'un corps articulé peut être décrite par la même équation que pour les corps rigides, même si Φ et $\mathbf{b}$ ont des expressions plus compliquées.

Différentes approches pour simuler un système moléculaire existent. Nous les décrivons dans la section suivante.

1.2 Simulation de systèmes moléculaires

Lors d'une simulation moléculaire, nous ne sommes pas intéressés par une trajectoire particulière du système dans l'espace des phases, mais par les propriétés moyennes de ce système (par exemple la température). Il y a plusieurs raisons pour cela. Tout d'abord, les valeurs moyennes obtenues avec la simulation sont faciles à comparer avec des résultats expérimentaux, ce qui n'est pas vrai, par exemple, pour les valeurs exactes des positions ou moments des particules. Ensuite, si nous simulons l'évolution du système moléculaire en temps, même une petite différence dans les conditions initiales conduira à une différence exponentielle dans les trajectoires au fil du temps [23, 24].

Les propriétés moyennes d'un système moléculaire sont normalement étudiées dans un ensemble de configurations (ou d'états) du système qui correspondent à plusieurs critères. L'ensemble des états possibles correspondant à certains critères s'appelle un ensemble statistique. Les ensembles les plus fréquemment utilisés sont les suivants:

- l'ensemble **microcanonique** qui décrit le comportement d'un système isolé thermiquement (ensemble NVE, le nombre de particules N, le volume V et l'énergie E sont fixes);
- l'ensemble **canonique** qui a une température T bien définie (ensemble NVT, N et V sont aussi des constantes);

- l'ensemble **isobarique-isothermique** (NPT, un ensemble à température et pression constantes);
- l'ensemble **grand-canonique** (μVT, ou μ est un potentiel chimique)[1].

Les deux techniques standard pour effectuer des simulations moléculaires sont la dynamique moléculaire (DM) et la méthode de Monte Carlo (MC) [23]. Ces deux approches sont très différentes. La dynamique moléculaire consiste en l'intégration des équations du mouvement du système pendant une certaine période de temps, et la moyenne des propriétés est faite au fil du temps. La méthode de Monte Carlo peut utiliser un certain nombre de stratégies différentes pour générer des configurations du système, sur lesquelles les propriétés sont moyennées. L'"hypothèse ergodique" indique que, dans un calcul de statistiques, une moyenne sur une période suffisamment longue (DM) est équivalente à la moyenne sur l'ensemble (MC). Cette affirmation n'est pas toujours vraie en pratique. Ainsi, selon le problème, la technique la plus appropriée doit être choisie.

Les simulations de DM et MC alternent une ou plusieurs itérations de deux étapes principales: (i) calculer les forces et les énergies pour la configuration actuelle du système, et (ii) avec ces informations, générer la configuration suivante. Nous allons maintenant décrire ces deux étapes en détail.

1.2.1 L'énergie potentielle et les forces

Lors d'une simulation, un modèle doit être choisi pour représenter les interactions dans le système. Précisément, une fonction d'énergie potentielle (ou un champ de force) doit être précisée. De nombreux champs de force existent [26], obtenus à partir de calculs ab initio et de différents types de données expérimentales. Une forme de base d'un champ de force classique contient des termes liés, décrivant la connectivité des structures chimiques dans le système (par exemple des liaisons ou des angles de liaison) et des termes non-liés, précisant des interactions à courte portée (*e.g.* van der Waals) et à longue portée (*e.g.* électrostatique).

Les termes liés et non-liés sont souvent considérés comme ne dépendant que de quelques degrés de liberté, et dans le cas le plus simple sont réduits à des interactions paires. Pour les interactions de van der Waals, on considère souvent que le potentiel diminue rapidement, de sorte que les particules ne sont pas censées interagir si la distance qui les sépare dépasse une valeur prédéfinie – *distance de coupure*.

Les forces d'interaction peuvent être obtenues par le calcul du gradient de la fonction de potentiel V:

$$\mathbf{F} = -\nabla V.$$

Comme chaque particule dans le système interagit souvent avec plusieurs autres particules, les calculs de forces représentent une partie importante de tous les calculs effectués lors d'une simulation.

[1] La technique de "l'ensemble" de Gibbs [25] est souvent utilisée pour obtenir le comportement des phases de fluides et de mélanges.

1.2.2 Génération de configurations

Des stratégies différentes sont utilisées par la dynamique moléculaire et les méthodes de Monte Carlo pour générer des configurations du système.

1.2.2.1 Dynamique moléculaire

Dans une simulation de dynamique moléculaire, les équations différentielles du mouvement doivent être intégrées numériquement dans le temps pour obtenir l'état du système lors de l'étape suivante. En général, l'état suivant y_{n+1} d'un système est une fonction de son état actuel y_n et du pas de temps h: $y_{n+1} = F(y_n, h)$. Par exemple, si nous discrétisons l'équation différentielle $\dot{y} = f(y)$ par la méthode des différences finies $\dot{y} = (y_{n+1} - y_n)/h$, le schéma d'intégration peut être $y_{n+1} = y_n + f(y_n)h$.

1.2.2.2 Monte Carlo

Les simulations de Monte Carlo peuvent utiliser plusieurs méthodes pour générer une configuration d'essai d'un système (par exemple déplacer une ou plusieurs particules de manière aléatoire), qui est *acceptée* ou *rejetée* avec une probabilité qui dépend de cette configuration d'essai et de l'état du système actuel [27, 28, 29, 25].

1.3 Simulations efficaces

Les systèmes moléculaires présentant une utilité en physique, chimie et biologie sont souvent constitués d'un grand nombre d'atomes. Par exemple, un ribosome (une partie importante de la cellule, qui produit des chaines protéiques) se compose d'environ 100 000 atomes, et une capside de virus peut contenir jusqu'à plusieurs millions d'atomes. La simulation d'un tel système pendant une longue période (nécessaire pour observer certains phénomènes), ou la collecte de statistiques complexes sur elle, devient un calcul coûteux en temps machine.

Dans cette section, nous examinons plusieurs approches (théorique et numérique) qui ont été développées pour effectuer des simulations moléculaires efficaces, c'est à dire des simulations qui montrent la meilleure performance pour un problème donné. Pour une simulation efficace, la représentation du système (résolution, modèle d'interaction, système de coordonnées, etc) ainsi que la méthode de simulation et de mise en œuvre doivent être soigneusement choisies.

1.3.1 Choix d'un modèle approprié

Le choix du modèle pour le système affecte à la fois la performance de la simulation et l'ensemble des propriétés qui peuvent être récupérées à l'issue du calcul.

Examinons quelques représentations possibles des systèmes moléculaires et leurs domaines d'application.

1.3.1.1 Représentation tout-atome

Cette représentation prend en compte tous les atomes dans un système moléculaire.

Dans ce cas, les simulations sont souvent effectuées dans l'espace cartésien: les coordonnées des atomes sont propagées dans le temps, par l'équation de Newton. Les coordonnées cartésiennes sont faciles à définir et sont naturelles pour des interactions non-liées et à longue portée. Cependant, les interactions liées sont décrites par des contraintes ou des termes d'énergie [30, 31].

Il existe une grande variété de champs de force empiriques pour cette représentation. Ces champs de force sont différents, d'abord, dans la forme fonctionnelle des termes liés et non-liés. Par exemple, dans certains champs de force, des liaisons covalentes sont modélisées par des oscillateurs harmoniques et ne peuvent pas être rompues lors de la simulation, mais il existe aussi quelques champs de force réactifs [32, 33] qui permettent la rupture des liaisons. En plus, les champs de force utilisent différents ensembles de paramètres pour chaque atome (masse, charge, rayon de van der Waals, etc), ainsi que des valeurs d'équilibre pour, par exemple, les longueurs et les angles des liaisons. En conséquence, il existe des champs de force généraux, comprenant tous les types d'atomes [34], et des champs de force spécifiques, conçus pour quelques types d'atomes uniquement [35, 33].

Il existe aussi des expressions *ab initio* d'énergie potentielle, basées sur les principes de la mécanique quantique. Ces expressions sont coûteuses à évaluer et sont limitées aux petits systèmes. Pour combiner les avantages des potentiels ab-initio et empiriques, des approches mêlant les deux ont été créées (par exemple, la technique QM / MM [36]).

1.3.1.2 Représentation des atomes unifiés

Avec cette représentation, des atomes d'hydrogène sont parfois unis avec d'autres atomes pour former de nouveaux sites d'interaction. Cela réduit le nombre de particules dans le système et permet de simuler des systèmes plus grands.

Pour cette représentation, plusieurs champs de force ont été proposés [37, 38], y compris des champs de force transférables (par exemple TRAPPE [39]) : les paramètres d'interaction pour un site donné restent les mêmes pour les différentes molécules, et le champ de force est transférable entre différents états du système (température, pression, etc.)

1.3.1.3 Représentation atomistique, réduction du nombre de degrés de liberté

Une autre façon de réduire le nombre de degrés de liberté dans le système consiste à fixer certains d'entre eux (par exemple les longueurs de liaison) à des valeurs spécifiques.

Pour cela, des simulations en coordonnées internes [40, 41, 42, 43, 44, 45, 46, 47] sont naturelles : dans ces simulations, les longueurs et angles de liaison, ainsi que les angles de torsion, représentent directement les degrés de liberté et peuvent être facilement fixés à certaines valeurs[2]. Le nombre de degrés de liberté dans le système est, par conséquent, considérable-

[2] D'habitude les vibrations des liaisons sont de haute fréquence mais de faible amplitude. Ces liaisons donc

ment diminué (plus précisément, fixer les longueurs et angles de liaison dans une simulation en coordonnées internes peut diviser le nombre de variables par six ; et imposer en plus une géométrie plane covalente – par dix [48]), des pas de temps plus grands sont autorisés [49] (comme les vibrations à haute fréquence pour les liaisons ne sont plus prises en compte) et un échantillonnage plus rapide est possible [46]. La mécanique moléculaire en coordonnées internes s'est révélée être utile dans plusieurs applications, y compris la modélisation moléculaire, l'assemblage moléculaire [44], la RMN [43], et est aussi employée par les algorithmes diviser-pour-régner récursifs [21, 22].

Néanmoins, l'application directe des champs de force standards aux simulations avec des contraintes (notamment, aux simulations en coordonnées internes avec des liaisons et des angles fixes) est discutable, et une "projection" de tels champs de force pourrait être nécessaire [48]. Cette "projection" est une tâche très difficile à réaliser, et tous les paramètres d'interaction sont transférés uniquement pour une valeur fixe de température.

1.3.1.4 Représentation gros-grain et DPD

Dans le but de réduire considérablement le nombre de particules dans le système, des "pseudo-atomes" peuvent être utilisés pour représenter des groupes d'atomes (molécules d'eau par exemple), ce qui produit une représentation gros-grain [50, 51, 52].

Dans ce cas, de nouveaux modèles d'interaction (avec des potentiels qui échantillonnent correctement l'espace de phase) doivent être développés, ce qui est un travail long et compliqué [53, 54]. Ensuite, il peut être difficile de porter un champ de force à gros grains spécifique d'un logiciel de simulation à un autre. En plus, les champs de force à gros grains sont généralement obtenus à partir des champs de force tout-atome à une température fixe, et, par conséquent, ne peuvent être utilisés que dans les simulations à cette température particulière. Cependant, récemment, le champ de force à gros grains UNRES [55] a été introduit. L'expression de sa fonction potentielle intègre des termes dépendants de la température, et il permet de calculer les propriétés thermodynamiques aux différentes températures.

Pour la dynamique des fluides, la dynamique des particules dissipatives (DPD) [56, 57] a été développée. Dans cette approche, les particules représentent des molécules ou des régions de fluide.

1.3.1.5 Représentations hybrides

Récemment, des méthodes basées sur des représentations multi-résolution ont été proposées pour coupler les niveaux tout-atome et gros-grain de la description du système [58, 59, 60, 61, 62, 63].

Dans ces méthodes, à l'échelle atomique les calculs sont exacts, mais lents. Habituellement, un petit sous-système d'un grand intérêt est représenté à cette échelle. D'autre part, dans la partie gros-grain de la simulation, la précision est faible, mais les calculs s'effectuent plus rapidement. Les techniques les plus intéressantes sont décrites dans les Réfs [60, 61, 62, 63].

---

peuvent être facilement modélisées comme fixées à leurs valeurs d'équilibre.

Ces méthodes sont basées sur l'interpolation des forces, des potentiels ou des lagrangiens entre les deux résolutions, à travers une région de transition. Cependant, la combinaison de niveaux de représentation différents est un problème difficile. Par conséquent, une autre approche intéressante est la méthode multi-grains [64] : un algorithme pour la simulation grain-fin et gros-grain simultanée des systèmes moléculaires.

Toutes les méthodes ci-dessus sont des méthodes de la mécanique classique.

Cependant, la mécanique classique des particules ou des corps rigides peut être combinée avec d'autres échelles. Par exemple, il existe des méthodes éléments finis[65]/dynamique moléculaire/mécanique quantique[66, 67, 68], une méthode hybride entre la dynamique moléculaire et la méthode Lattice Boltzmann [69], ainsi qu'une combinaison entre la méthode de fluctuation hydrodynamique [70] et de dynamique moléculaire dans Réf. [71]. Des modèles continus comme des modèles de solvants implicites [72] ou membranes implicites [73] peuvent être couplés avec des représentations plus fines.

Une fois que la représentation d'un système moléculaire est choisie, une méthode de simulation appropriée doit être déterminée.

1.3.2 Calculs d'énergie et de forces

Comme indiqué précédemment, le calcul des forces représente une partie importante du temps global de simulation. Effectuer un calcul de forces efficace est donc très important.

1.3.2.1 Approximation des forces

La complexité du calcul de la mise à jour des forces peut être améliorée en utilisant des approximations du potentiel, et, par conséquent, des forces. Par exemple, la méthode multipôle rapide (FMM) [74] réduit le coût du calcul des interactions à longue portée de $O(N^2)$ à $O(N)$ pour un système de N particules. Une autre approche souvent utilisée est la méthode du maillage des particules d'Ewald (Particle-Mesh Ewald method) [75] qui réduit le même coût de $O(N^2)$ à $O(N \log N)$.

1.3.2.2 Mise à jour partielle

Parfois, seulement certaines forces du système doivent être mises à jour. Le temps de cette mise à jour peut donc être amélioré, par exemple pour les systèmes présentant des symétries [76], ou pour les systèmes dont les termes d'interactions à longue et courte portée sont séparés [77], auquel cas les forces à longue portée peuvent être mises à jour non pas à chaque pas de temps, mais tous les n pas.

Les algorithmes incrémentaux peuvent aussi accélérer la mise à jour des forces. Ces algorithmes mettent à jour les propriétés du système en partant de sa configuration précédente et en utilisant les informations sur les changements qu'il a subis, au lieu de les recalculer de zéro.

Récemment, des algorithmes adaptatifs ont été introduits [78, 79, 41, 80]. Ces algorithmes sont principalement utilisés pour la simulation adaptative et quasi-statique des corps articulés.

Ils permettent de choisir le nombre de degrés de liberté actifs à chaque pas de temps (et ce faisant, la précision de la simulation), c'est à dire effectuer une mise à jour partielle de l'état du système à chaque pas de temps. Le compromis entre précision et temps de calcul peut alors être arbitré de manière lisse, ce qui permet de diminuer significativement le nombre de degrés de liberté tout en gardant la précision nécessaire à la description du mouvement, d'où des accélérations de simulation importantes.

1.3.3 Génération de configurations

Une méthode de simulation appropriée doit être choisie en fonction de l'application.

Plusieurs auteurs ont comparé l'efficacité de la dynamique moléculaire et les méthodes de Monte Carlo pour des applications spécifiques : simulations de protéines (DM a été plus efficace dans la Réf. [81]), simulations de liquides (MC a été plus rapide dans la Réf. [82]), simulations de repliement des protéines (MC a été plus rapide dans la Réf. [83]), etc.

Aujourd'hui cependant, la dynamique moléculaire et la méthode de Monte Carlo sont souvent combinées pour effectuer des simulations efficaces : parfois, des pas de temps dynamiques sont utilisés avec Monte Carlo, où un échantillonnage intelligent de l'espace des phases est réalisé à l'aide d'une ou plusieurs simulations de dynamique moléculaire [84, 85, 86].

Dans ce qui suit, nous allons continuer d'associer l'intégration numérique à la dynamique moléculaire, et de l'échantillonnage intelligent à la méthode de Monte Carlo, en n'oubliant pas que pour une simulation efficace nous pourrions avoir besoin des deux.

1.3.3.1 Dynamique moléculaire et intégration efficace

Comme il est expliqué ci-dessus, la génération des configurations nouvelles dans une simulation de dynamique moléculaire demande l'intégration des équations du mouvement dans le temps.

De nombreux schémas d'intégration possibles ont été proposés [87].

- Les schémas implicites et explicites peuvent être distingués : pour déterminer l'état d'un système à l'étape suivante, les intégrateurs explicites utilisent uniquement l'état actuel, alors que les méthodes implicites doivent résoudre une équation impliquant les états actuel et suivant du système. Les intégrateurs explicites sont les plus simples, mais pour certains problèmes les méthodes implicites sont préférables.

- L'ordre de l'intégrateur joue un rôle important. Plus il est grand, plus la solution au problème est précise. En contrepartie, toutes les dérivées temporelles des coordonnées de particules, jusqu'à l'ordre de l'intégrateur, doivent être calculées et stockées.

- Il existe des intégrateurs symplectiques qui préservent la structure géométrique du flux hamiltonien [88]. Ces intégrateurs bornent les fluctuations d'énergie et permettent de conserver l'énergie pendant de longues périodes pour les simulations en ensemble NVE.

- Certains intégrateurs sont réversibles dans le temps (c'est le cas des équations du mouvement de Newton). Une approche générale pour construire un intégrateur réversible

utilise la factorisation de Trotter du propagateur de Liouville [77]. Des expériences numériques montrent que certaines méthodes symétriques (réversibles) présentent des propriétés similaires à celles des méthodes symplectiques pour certains systèmes [87].

- On oppose aussi les méthodes mono-étape aux méthodes multi-étapes. Dans le premier cas, l'état du système à l'étape suivante ne dépend que de son état à l'étape précédente; dans le second, il dépend de plusieurs étapes précédentes.

Le choix du pas d'intégration a une importance cruciale pour la méthode de simulation. En pratique, le pas de temps doit être court par rapport à la plus petite période de vibration des particules. Normalement, il est choisi environ 10 fois plus petit que la période de vibration la plus courte dans la simulation. D'autre part, pour l'efficacité de la simulation, le pas de temps doit être choisi le plus grand possible.

Il existe des méthodes qui augmentent le pas de temps [89, 90, 91, 92, 93, 94], permettant ainsi d'accélérer les simulations et d'utiliser de plus grandes échelles de temps. Ceci peut être réalisé, par exemple, en séparant les degrés de liberté lents et rapides dans le système [89] (dans le cas général [77]), ou en changeant le tenseur de masse [90, 91, 94], ou encore en utilisant un pas de temps adaptatif [92], etc.

Le pas de temps de la simulation peut également être augmenté en fixant les liaisons dans le système pour supprimer les modes de haute fréquence. Cette idée est utilisée par exemple dans les méthodes SHAKE [30] et RATTLE [31] (ainsi que leurs nombreuses variantes), dans la méthode de la fonction de pénalité [95], et dans les méthodes qui modifie le potentiel [96].

1.3.3.2 Monte-Carlo et échantillonnage efficace

Une grande variété de techniques visant à accélérer l'échantillonnage d'espace des phases a été développée [97, 85, 98].

Par exemple, plusieurs modifications de la méthode traditionnelle de Monte-Carlo existent [99] [29] [100, 101, 102]. Pour les évènements rares, la méthode de Monte Carlo cinétique peut également être utilisée [103]. Certains algorithmes comme Monte Carlo hybride [84, 104] essaient de combiner les avantages de DM et MC.

Certaines méthodes d'échantillonnage utilisent des simulations de dynamique moléculaire. Par exemple, dans l'approche d'"échange des répliques" [85] plusieurs copies du système d'origine sont simulées sans interactions entre elles, et à des températures différentes. Elles sont régulièrement échangées avec une certaine probabilité qui dépend des énergies de chaque copie. Les moyennes statistiques dans ce cas sont obtenues à partir des copies du système à la température désirée. Cette méthode est largement utilisée pour les simulations du repliement des protéines.

Avec la méthode LES [98] un petit sous-système d'intérêt est clôné, résultant en plusieurs copies sans interaction entre elles, et le reste du système subit le potentiel moyen de ces copies. L'espace de phase est donc mieux échantillonné qu'avec la DM traditionnelle.

La dynamique moléculaire accélérée [105, 97] permet d'augmenter l'échelle de temps des simulations d'évènements rares en modifiant le potentiel.

La dynamique moléculaire guidée (Steered molecular dynamics, SMD) [106] est une méthode qui permet d'observer les processus biologiques à des échelles de temps accessibles par des simulations moléculaires. L'idée générale de cette méthode consiste à appliquer des forces extérieures sur un ou plusieurs atomes du système pour étudier sa réponse dans des conditions variées. Par exemple, une interaction protéine-ligand peut être étudié en tirant sur le ligand au niveau du site de liaison.

Les calculs d'une classe spécifique de propriétés thermodynamiques (celles qui ne peuvent pas être obtenues par une simple moyenne d'une fonction des coordonnées et des moments sur toutes les particules), par exemple l'énergie libre de Helmholtz et ses différences entre les divers états, ont été largement étudiés [107]. En particulier, les calculs d'énergie libre de solvatation [19, 108, 109] sont d'un grand intérêt, étant donné que souvent le solvant représente une part indisponsable d'un système moléculaire.

1.3.4 Matériel informatique

Nous discutons maintenant de l'accélération des simulations moléculaires grâce à des techniques matérielles.

Si un problème peut être facilement subdivisé en plusieurs petits problèmes qui peuvent être résolus simultanément, le calcul parallèle est une technique d'accélération naturelle : les calculs sont répartis entre plusieurs unités de traitement, puis les résultats sont réunis pour fournir le résultat final. La première façon d'effectuer des simulations moléculaires en parallèle est de séparer les calculs d'une seule simulation. Cette stratégie peut être appliquée à des méthodes comme diviser-pour-régner [21] ou la décomposition de domaine [110, 111]. Une autre option consiste à effectuer plusieurs simulations en parallèle, par exemple, la méthode d'"échange des répliques" [85].

Le matériel moderne offre de nombreuses possibilités pour le calcul parallèle. Par exemple, les processeurs multicœurs (CPU) sont maintenant très courants, et ce même pour les ordinateurs de bureau. Les unités de calcul graphiques peuvent aussi accélérer les simulations à N corps [112, 113] ou les simulations de Monte Carlo [114]. Des calculs importants peuvent être effectués sur des clusters ou des super-ordinateurs [115]. Il existe même des super-ordinateurs spécialement conçus pour les simulations moléculaires [116, 117]. Les systèmes distribués, qui se composent d'un grand nombre d'ordinateurs autonomes qui interagissent à travers un réseau, peuvent aider à résoudre des problèmes importants, tels que le repliement des protéines [118].

1.3.5 Logiciels

Plusieurs des méthodes abordées ci-dessus, combinées avec d'autres techniques d'optimisation, sont intégrées dans une variété de champs de forces sophistiqués et de logiciels de simulation qui ont été écrits pour effectuer des simulations classiques ou quantiques, par exemple CHARMM [119], AMBER [120], ABINIT [121], Desmond [122], NAMD [123], GROMOS [124] et GROMACS [125], LAMMPS, SIESTA, VASP, TINKER, YASARA, etc.

Certains de ces outils sont libres de droits, tandis que d'autres sont disponibles dans le commerce, parfois via l'intégration dans des applications: Discovery Studio, Matériaux Studio, Médée, ME, Designer Ascalaph, BOSS, Spartan, Maestro, NanoEngineer-1, etc. D'autres outils sont essentiellement dédiés à la visualisation, mais peuvent parfois être liés aux logiciels de simulation: VMD [2], PyMol [1], Zodiac, Avogadro, etc. Le réseau nanoHUB [126] comprend également un ensemble complet d'outils de calcul liés à la nanoscience.

1.4 Contributions

L'objectif principal de cette thèse était de contribuer à l'accélération des simulations moléculaires. Précisément, nous avons considéré les algorithmes qui permettent de réduire le nombre de degrés de liberté dans le système, tout en restant à la résolution des particules pour calculer l'énergie potentielle et les forces. Dans ces algorithmes, certains blocs d'atomes se déplacent ensemble en tant que corps rigides, ce qui réduit le nombre de forces à mettre à jour à chaque pas de temps, puisque les forces à l'intérieur des corps rigides sont constantes et ne doivent pas être mises à jour. Comme les calculs de force représentent une partie importante des calculs de simulation moléculaire, cette approche peut résulter en une accélération significative.

Le processus d'évaluation des forces consiste typiquement en deux parties. Tout d'abord, la recherche des voisins est effectuée : toutes les paires dont les atomes sont plus proches entre eux que la distance de coupure sont considérées en interaction (pour plus de simplicité, nous considérons les potentiels paires). Ces paires sont répertoriées et stockées dans une liste de voisins. Ensuite, les forces correspondant à chaque paire de voisins sont calculées. Il se trouve que si les grands blocs d'atomes se déplacent ensemble, l'étape de recherche des voisins peut être accélérée. Dans la première partie de notre travail, nous présentons donc une étude qui compare différentes méthodes pour effectuer une recherche de voisinage entre grands groupes rigides d'atomes. Plus précisément, nous comparons l'algorithme traditionnel basé sur une grille à une série d'algorithmes qui utilisent des volumes englobants pour éliminer rapidement les grands groupes de paires d'atomes non pertinents lors de la recherche de voisins. Ces derniers ont été utilisés en géométrie algorithmique, avec des applications en infographie, robotique, réalité virtuelle, etc. Nous comparons les performances de ces algorithmes en variant plusieurs paramètres : la taille des molécules, la distance moyenne entre elles, la distance de coupure, ainsi que le type de volume englobant utilisé dans la hiérarchie (AABB, OBB, 2 types de sphères). Nous démontrons que pour les systèmes de taille relativement grande (>100000 atomes) l'algorithme fondé sur la hiérarchie des sphères montre les meilleurs résultats, et l'algorithme traditionnel, utilisant la grille, donne les pires. Pour les petits systèmes, cependant, l'algorithme avec une grille et celui de la hiérarchie de sphères sont les plus efficaces. Ces résultats ont été publiés dans le *Journal of Computational Chemistry* [127].

Ayant remarqué que les approches hiérarchiques sont souvent bénéfiques pour les simulations où les grands blocs d'atomes pourraient être considérés comme des corps rigides, nous avons découvert qu'aucune méthode générale de construction d'une telle hiérarchie n'avait été proposée. C'est pourquoi, dans la deuxième partie de cette thèse, nous présentons un algo-

rithme rapide et général pour la construction complète d'une représentation hiérarchique d'un système moléculaire. Cet algorithme en trois étapes traite le système d'entrée sous forme d'un graphe moléculaire dans lequel les sommets représentent des atomes ou des pseudo-atomes, et les arêtes représentent des liaisons covalentes. Nous démontrons la performance de notre algorithme sur un ensemble de cas difficiles, ainsi que sur un grand sous-ensemble de graphes moléculaires extrait de la Protein Data Bank [128]. Finalement, nous démontrons une application de notre algorithme à la mécanique moléculaire adaptative en angles de torsion. Ces résultats ont également été publiés dans le *Journal of Computational Chemistry* [129].

Les méthodes adaptatives basées sur des degrés de la liberté figés comme [78, 79, 41, 80] n'ont jamais été complétées par une analyse des propriétés des simulations obtenues (par exemple leur stabilité). Pour les applications en infographie ce n'est pas important, puisque les simulations doivent être visuellement plausibles, mais pas nécessairement théoriquement rigoureuses. Dans les simulations de dynamique moléculaire, cependant, il est crucial d'analyser chaque nouvelle approche et de vérifier si elle produit des simulations qui échantillonnent correctement l'espace des phases.

C'est pourquoi, dans la troisième partie de cette thèse, nous présentons une nouvelle méthode mathématiquement fondée et basée sur la dynamique hamiltonienne, qui active et désactive des degrés de liberté en position, ce qui réduit le nombre de forces mis à jour à chaque pas de temps, et peut considérablement accélérer les simulations. Cette méthode est adaptée pour une classe plus générale de simulations qui est celle des simulations de particules, et nous l'appelons donc ARPS: Simulations de Particules Restreintes de façon Adaptative (*ARPS: Adaptively Restrained Particle Simulations*). Notre approche présente de nombreux avantages. Par exemple, elle produit des simulations longues et stables. Pour des simulations à énergie constante, elle permet aux utilisateurs de choisir de manière arbitraire et lisse le compromis entre précision et vitesse de calcul, et ainsi d'obtenir rapidement des trajectoires approximatives. Ce compromis entre précision et vitesse de calcul peut être choisi pour chaque particule indépendamment, afin que les utilisateurs puissent concentrer ARPS sur des régions spécifiques du système simulé (par exemple un polymère dans du solvant). Un autre avantage important est que, lors de l'exécution d'ARPS dans l'ensemble canonique (NVT), les propriétés statiques d'équilibre correctes peuvent être récupérées. Nous proposons une approche générale fondée sur une modification de la matrice d'inertie inverse dans le hamiltonien, ainsi qu'un choix particulier de cette matrice pour effectuer des simulations en coordonnées cartésiennes, où des degrés de liberté en position des particules sont activés et désactivés de manière indépendante. Pour effectuer des simulations efficaces, nous proposons des algorithmes incrémentaux pour la mise à jour des forces. Nous illustrons ARPS sur plusieurs expériences numériques, notamment (a) un exemple de cascade de collisions qui montre comment ARPS permet de choisir finement le compromis entre précision et vitesse de simulation, et (b) une étude de polymère solvaté qui montre comment il est possible d'obtenir rapidement des propriétés statiques d'équilibre avec ARPS. Ces résultats ont été publiés dans *Physical Review Letters [130]* et un brevet a été déposé.

Une façon naturelle de procéder était, alors, de développer une approche hiérarchique

pour ARPS. Dans le dernier Chapitre de cette thèse, nous présentons une telle approche. Précisément, nous introduisons un autre choix de la matrice d'inertie inverse et les simulations correspondantes, approche appelée "Simulations hiérarchiques de particules restreintes de façon adaptative" (ARPS hiérarchiques). Dans ce cas, nous activons et désactivons des degrés de liberté relatifs du système (c'est à dire, nous groupons les particules dans les corps rigides), et non pas simplement les degrés de liberté des particules individuelles. Un brevet est en train d'être rédigé sur ce sujet.

Pour résumer, dans cette thèse nous proposons plusieurs nouveaux algorithmes généraux qui accélèrent les simulations moléculaires ainsi qu'une comparaison des méthodes existantes pour trouver la plus adaptée à des simulations spécifiques. Nous fournissons des méthodes qui sont conçues exprès pour certains types de simulations, ainsi qu'une approche générale pour effectuer des simulations de particules.

Cette thèse est organisée de la façon suivante. Tout d'abord, dans le Chapitre 2, nous présentons l'étude comparative des méthodes de recherche des voisins pour les grandes molécules rigides. Puis, dans le Chapitre 3, nous présentons un nouvel algorithme visant à construire une représentation hiérarchique d'un graphe moléculaire général. Plus tard, dans le Chapitre 4, nous décrivons une approche totalement nouvelle pour accélérer les simulations moléculaires (et, plus généralement, des particules) en activant et désactivant des degrés de liberté en position du système lors une simulation (ARPS). Puis, dans le Chapitre 5 nous proposons une approche hiérarchique de cette nouvelle méthode (ARPS hiérarchiques). Enfin, au Chapitre 6 nous décrivons les conclusions et des perspectives sur les travaux futurs.

Chapter 1

Introduction

Molecular simulations play an increasingly important role in investigating matter at the nanoscale. They allow to understand the nature of many chemical and biological processes in various materials [3, 4] and in a living cell [5, 6], to observe new phenomena and discover new characteristics of physical matter [7] and to predict the behavior of systems of interest [8]. Molecular simulations may also help in testing and exploring new physical models, and in explaining poorly understood phenomena. They can be used to model and design new molecules with desired properties [9], to perform simulations under conditions and states that are hard to create in a laboratory [10] while easily tuning the system's parameters. Thus, molecular simulations are nowadays widely used in many research fields, *e.g.* computational chemistry, physics and biology, material science. The obtained results can have direct applications to such important technologies as drug design [11, 12, 13, 14], nanotechnology [15], electronics [16], optics, etc.

Unfortunately, many important problems in chemistry, biology and physics, *e.g.* protein folding [17, 18], molecular docking [5, 131, 132], molecular solvation [19], diffusion across bio-membranes [20], simulating chemical reactions [19], etc. still remain challenging. This dissertation describes several new algorithms aimed at accelerating molecular simulations.

A molecular simulation consists in creating a computer representation (model) for the molecular system and, then, generating a number of realistic configurations of this system. From these configurations, different types of information can be extracted: thermodynamical properties, dynamical properties, structural particularities, etc.

Let us first discuss possible representations of molecular systems.

1.1 Representing a molecular system

There exist several ways to represent a molecular system. Quantum mechanics, for example, considers this system as a collection of atoms and subatomic particles, and describes their properties by a wavefunction. On a larger scale, atoms, groups of atoms, or molecules may be represented as discrete objects interacting through some (empirical) potential function.

Finally, a system might be modeled as a continuous mass: fluid or solid.

In this dissertation, we focus our attention on the discrete representation of molecular systems subject to classical mechanics, namely particles, rigid bodies and articulated bodies, and consider their dynamics.

1.1.1 Particle dynamics

A molecular system in 3D might be considered as a collection of N point-mass particles (*e.g.* atoms). One way to describe the behavior of such system is through a Hamiltonian function $H(\mathbf{q}, \mathbf{p})$, where $\mathbf{q}$ and $\mathbf{p}$ are $3N$-vectors of particles positions and momenta respectively. A Hamiltonian is usually expressed as a sum of kinetic and potential energies of the system: $H = K + V$. The precise form of these energies, however, is chosen according to the system under study. A widely-used form of the Hamiltonian is:

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

where $\mathbf{M}$ is a $3N \times 3N$ diagonal mass matrix (which might depend on positions $\mathbf{q}$), and $V(\mathbf{q})$ is an interaction potential.

The equations of motion can be derived from this Hamiltonian:

$$\begin{aligned}\dot{\mathbf{p}} &= -\frac{\partial H(\mathbf{q}, \mathbf{p})}{\partial \mathbf{q}} = -\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}}, \\ \dot{\mathbf{q}} &= \frac{\partial H(\mathbf{q}, \mathbf{p})}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p}.\end{aligned}$$

1.1.2 Rigid-body/Articulated-body dynamics

Rigid bodies (in contrast with particles) occupy some volume in space and have geometrical properties as, *e.g.* a center of mass and moments of inertia. They are considered to be not deformable. While particles perform only translational motion in the directions of the coordinate axes, 3D rigid bodies have 6 degrees of freedom (translation and rotation in the directions of axes).

To write the equation of motion for a rigid body, it is convenient to use spatial notations [21]. These notations describe rigid-body velocity, acceleration, inertia etc., using 6D vectors and tensors. The equation of motion of a rigid body can then be written as follows:

$$\mathbf{f} = \frac{d}{dt}(\mathbf{I}\mathbf{v}) = \mathbf{I}\mathbf{a} + \mathbf{v} \times \mathbf{I}\mathbf{v},$$

where $\mathbf{f}$ is the spatial force acting on the rigid body, $\mathbf{I}$ is the spatial inertia of the rigid body, $\mathbf{v}$, $\mathbf{Iv}$, and $\mathbf{a}$ are its spatial velocity, momentum, and acceleration respectively, and "$\times$" stands for a spatial cross-product. Hence, the acceleration of a rigid body may be written:

$$\mathbf{a} = \mathbf{I}^{-1}\mathbf{f} - \mathbf{I}^{-1}(\mathbf{v} \times \mathbf{I}\mathbf{v}) = \Phi\mathbf{f} + \mathbf{b},$$

where Φ is the spatial inverse inertia of the rigid body, and $\mathbf{b}$ is its bias acceleration (*i.e.* the acceleration it has when no force is applied to it).

Articulated bodies, also widely-used in robotics and computer graphics, are rigid bodies with constraints. It has been shown [21, 22] that the dynamics of an articulated body can be described by the same equation as for rigid bodies, although Φ and $\mathbf{b}$ have more complex expressions.

Different approaches to simulate molecular systems exist. We describe them in the next section.

1.2 Simulating molecular systems

When performing a molecular simulation, we are not interested in one particular trajectory of the system in phase space, but in the system's average properties (*e.g.* temperature, heat capacity, etc.). There are several reasons for this. First, average values obtained from the simulation are easy to compare with experimental results, and this is not true for, *e.g.* exact values of positions or momenta of the particles. Second, if we simulate the evolution of the molecular system in time, even a small difference in initial conditions will result in an exponential difference in trajectories over time [23, 24].

Average properties of a molecular system are normally studied in a set of system's configurations (or states) that correspond to several criteria. The set of all possible states corresponding to these criteria is called a statistical ensemble. The most frequently-used ensembles are:

- **the microcanonical ensemble** that describes the behavior of a thermally isolated system (NVE ensemble, number of particles N, volume V and energy E are fixed);
- **the canonical ensemble** that has a well-defined temperature T (NVT ensemble, N, V and T are constant);
- **the isobaric-isothermal ensemble** (NPT, constant temperature and constant pressure ensemble);
- **the grand canonical ensemble** (μVT, where μ stands for chemical potential)[1].

Two standard techniques to perform molecular simulations are Molecular Dynamics (MD) and the Monte Carlo (MC) method [23]. These are two very different approaches. Molecular dynamics consists in integrating the equations of motion of the system for some period of

[1] The Gibbs "ensemble" technique [25] is also commonly used for obtaining the phase behavior of fluids and mixtures.

time, and averaging system's properties over time. The Monte Carlo method may use a number of different strategies to generate system configurations, over which the properties are averaged. The "ergodic hypothesis" states that, for computing statistics in a many-body system, averaging over a sufficiently long time (MD) is equivalent to the ensemble averaging (MC). However, this statement is not always true in practice. Thus, according to the problem, the most suitable technique should be chosen.

Both MD and MC simulations alternate one or several iterations of two major steps: (i) compute inter-particle forces and energies for the current system configuration, and (ii) based on this information, generate the next configuration. We now discuss these two steps in more details.

1.2.1 Potential energy and forces

To perform a simulation, a model should be chosen to represent interactions within the system. Precisely, a potential energy function (or a force field) should be specified. A large variety of force fields exist [26], obtained from ab initio calculations and different types of experimental data. A basic form of a classical force field contains bonded terms, related to the connectivity of chemical structures in the system (*e.g.* bonds or bond angles), and non-bonded terms, specifying short-range (*e.g.* van der Waals) and long-range (*e.g.* electrostatics) interactions.

Both bonded and non-bonded terms are often considered to only depend on a few relative degrees of freedom, and in the simplest case are reduced to pairwise interactions. For van der Waals interactions, the potential is often considered to be rapidly decaying, so that particles are assumed not to interact if the distance between them exceeds some pre-defined distance (also called *cutoff distance*).

Interaction forces can be obtained by computing the gradient of the potential function V:

$$\mathbf{F} = -\nabla V.$$

As each particle in the system usually interacts with several particles, force computations typically represent a significant part of all calculations performed during a simulation.

1.2.2 Generating configurations

Different strategies are used by molecular dynamics and Monte Carlo methods to generate system configurations.

1.2.2.1 Molecular dynamics

In a molecular dynamics simulation, differential equations of motion should be numerically integrated in time to obtain the state of the system at the next time step. In general, the next state y_{n+1} of a system is a function of its current state y_n and the time step h: $y_{n+1} = F(y_n, h)$. For example, if we discretize the differential equation $\dot{y} = f(y)$ by the finite difference method $\dot{y} = (y_{n+1} - y_n)/h$, then the integration scheme may be $y_{n+1} = y_n + f(y_n)h$.

1.2.2.2 Monte Carlo

Monte Carlo simulations may use various methods to generate a *trial* system's configuration (*e.g.* by randomly moving one or more particles), which is *accepted* or *rejected* with a probability that depends on this trial configuration and the current system's state [27, 28, 29, 25]. Random numbers generation plays an important role in Monte Carlo simulations, and numerous methods exist to produce pseudo-random numbers [133, 134, 135].

1.3 Efficient simulations

Molecular systems of interest in physics, chemistry and biology often consist of a large number of atoms. For example, a ribosome (an important part of the cell that produces protein chains) consists of around 100 000 atoms, and a virus capsid can contain up to several millions of atoms. Simulating such a system for a long time (necessary to observe certain phenomena), or collecting complex statistics on it, becomes computationally costly.

In this section, we discuss several approaches (both theoretical and numerical) that have been developed to perform *efficient* molecular simulations, *i.e.* simulations that show the best performance for a given problem. To run an efficient simulation, the representation of the system (resolution, interaction model, coordinate system, etc.) as well as the simulation method and its implementation should be carefully chosen.

1.3.1 Choosing an appropriate model

The choice of the model for the system affects both the performance of the simulation and the set of properties that can be retrieved from it.

Let us discuss some possible representations of molecular systems and their area of application.

1.3.1.1 All-atom representation

This representation takes into account every atom in a molecular system.

In this case, simulations are often carried out in Cartesian space: the coordinates of all atoms are propagated in time, following Newton's equation. Cartesian coordinates are easy to define and are natural for long-range non-bonded interactions. However, bonded interactions should be described as constraints or energy terms [30, 31].

There exists a large variety of empirical force fields for this representation. These force fields differ, first, in the functional form of the bonded and non-bonded terms. For example, in some force fields, covalent bonds are modeled with harmonic oscillators and may not be broken during simulation, but there also exist some *reactive* force fields [32, 33] that allow for bond breaking. Force fields also use different sets of parameters for each atom (mass, charge, van der Waals radius, etc.), as well as the equilibrium values for, *e.g.* bond lengths and angles.

As a result, there exist general force fields including all types of atoms [34], and specific force fields, designed for a few atom types only [35, 33].

There also exist ab-inito potential-energy expressions based on quantum mechanics principles [136]. However, these expressions are expensive to evaluate and are limited to small systems. To combine advantages of ab-initio and empirical potentials, approaches mixing them have been developed (for example, the QM/MM technique [36]).

1.3.1.2 United-atom representation

In this representation, in some atom groups, hydrogen atoms are united with other atoms into new interaction sites. This reduces the number of particles in the system and makes it possible to simulate larger systems.

For this representation, several force-fields have been proposed [37, 38], including transferable force fields (*e.g.* TraPPE [39]): their interaction parameters for a given site stay the same for different molecules, and the force field is transferable to different state points (temperature, pressure, etc.).

1.3.1.3 Atomistic representation, reduced number of degrees of freedom

Another way to reduce the number of degrees of freedom in the system is to fix some of them (*e.g.*, bond lengths) to specific values.

For that, simulations in internal coordinates [40, 41, 42, 43, 44, 45, 46, 47, 137] are natural: in these simulations, bond lengths, bond angles and torsion angles directly represent the degrees of freedom and may be easily fixed to some values[2]. The number of degrees of freedom in the system is, therefore, significantly decreased (precisely, simulations in internal coordinates with fixed bond lengths and bond angles can divide the number of free variables by six, and with additionally imposed planar covalent geometry – by ten [48]), larger simulation time steps are allowed [49] (as high-frequency vibrations for the bonds are no more considered) and faster conformational sampling is possible [46, 137]. Molecular mechanics in internal coordinates has been shown to be a useful tool in several applications, including molecular modeling, molecular docking [44], NMR [43], and is employed by the recursive divide-and-conquer algorithms [21, 22].

However, the direct application of standard force fields to simulations with constraints (namely, simulations in internal coordinates with fixed bonds and angles) is questionable and a "projection" of such force field might be needed [48]. This "projection" is a highly time-consuming and tedious task to perform, and all interaction parameters are then transferred only for a fixed temperature value.

[2] As usually bond vibrations are of high frequency but low amplitude, they can be easily modeled as fixed to their equilibrium values.

1.3.1.4 Coarse-graining and DPD

To significantly reduce the number of particles in the system, "pseudo-atoms" may be used to represent groups of atoms (*e.g.* water molecules), resulting in coarse-grained representations [50, 51, 52].

In this case, however, new interaction models (potentials that correctly sample phase space) have to be developed, and this is a complicated and time-consuming task [53, 54]. Then, it might be difficult to port a specific coarse-grained force field from one simulation package to another. Moreover, coarse-grained force fields are usually obtained from all-atom force fields by integrating out certain degrees of freedom at a fixed temperature, and, therefore, may only be used for simulations at this particular temperature. However, recently, the UNRES coarse-grained force field [55] was introduced. It has temperature-dependent terms in the potential function, and makes it possible to compute thermodynamic properties at several temperatures.

For fluid dynamics, dissipative particle dynamics (DPD) [56, 57] has been developed. In this approach, particles represent molecules or fluid regions, and an artificially-soft repulsive inter-particle potential is introduced.

1.3.1.5 Mixing representations

Recently, methods relying on multiresolution representations have been proposed to couple fine- (all-atom) and coarse-grained levels of system's description [58, 59, 60, 61, 62, 63].

In these methods, at the atomistic scale, calculations are accurate but slow. Usually, a sub-system of high interest is represented at this scale. On the other hand, in the coarse-grained part of the simulation, the accuracy is low, but the methods perform faster. The most interesting techniques are described in Refs. [60, 61, 62, 63]. These methods are based on interpolating forces, potentials or Lagrangians between the two resolutions, through a transition region. However, combining different levels of representation is a difficult problem. In particular, the so-called reverse-mapping problem arises when the coarse-grained representation should be mapped back to its fine-grained level. Therefore, another interesting approach is the multigraining method [64]: an algorithm for simultaneous fine-grained and coarse-grained simulation of molecular systems.

All the methods above are classical-mechanics methods.

However, classical mechanics of particles or rigid bodies may be also combined with other resolutions. For example, there exist Finite Element [65]/Molecular Dynamics/Quantum Mechanics methods [66, 67, 68], a hybrid Molecular dynamics/Lattice Boltzmann method [69], fluctuating hydrodynamics [70] combined with molecular dynamics in Ref. [71], etc. Continuum models as, *e.g.* implicit solvent models [72] or implicit membranes [73] may be coupled with finer representations.

Once the representation of a molecular system is chosen, a suitable simulation method should be determined.

1.3.2 Computing energies and forces

As mentioned before, computing forces represents a significant part of the overall simulation time. Performing efficient force computations is, therefore, very important.

1.3.2.1 Approximating forces

The computational complexity of the force update may be improved by using approximations of the potential function, and, therefore, of the particle forces. For example, the Fast Multipole Method (FMM) [74] reduces the cost of computing long-range interactions from $O(N^2)$ to $O(N)$ for a system of N particles. Another widely-used approach is the Particle-Mesh Ewald method [75] that reduces the same cost from $O(N^2)$ to $O(N \log N)$[3].

1.3.2.2 Partial update

Sometimes, not all forces in the system need to be updated. For example, the force update time may be improved for systems with symmetries [76], or for systems with separated long-range and short-range interactions [77], where long-range forces may be updated not at every step, but every n steps.

Incremental algorithms may also accelerate updating forces. These algorithms use computations from the previous system configuration and information about the changes in this system to update forces instead of recomputing them from scratch.

Recently, adaptive algorithms have been introduced [78, 79, 41, 80]. They are mainly used for adaptive simulation of articulated-body quasi-statics. These algorithms make it possible to arbitrarily choose the number of active degrees of freedom at each time step (equivalently, the precision of the simulation), *i.e.* perform a partial update of the system's state at each time step. The possibility of finely trading between precision and computational cost may result in significant speedups when a lower number of degrees of freedom is sufficient to describe the motion.

1.3.3 Generating configurations

An appropriate simulation method should be chosen depending on the application.

Several authors compared the efficiency of molecular dynamics and Monte Carlo methods for specific applications: protein simulations (MD was found to be more efficient in Ref. [81]), liquid simulations (MC performed faster in Ref. [82]), all-atom folding simulations (Monte Carlo was faster in Ref. [83]), etc.

Nowadays, however, molecular dynamics and Monte Carlo methods are often combined to perform efficient simulations: sometimes, dynamic steps are used in Monte Carlo simulations, or smart phase-space sampling is performed using one or several molecular dynamics simulations [84, 85, 86].

[3] Long-range interaction computations may additionally be accelerated by not recalculating them at all time steps but estimating them on the basis of linear prediction of time series [138].

In the following, we will continue associating molecular dynamics with integration and the Monte Carlo method with smart sampling, keeping in mind that for an efficient simulation we might need both.

1.3.3.1 Molecular dynamics and efficient integration

As explained above, new configurations in a molecular dynamics simulation are generated while integrating the equations of motion in time.

Many possible integration schemes have been proposed [87]:

- Implicit and explicit schemes can be distinguished: to determine the state of a system at the next step, explicit integrators only use its current state, while implicit methods solve an equation involving both the current and the next states. Explicit integrators are simpler, but for some problems implicit methods are preferable.

- The order of the integrator plays an important role. The higher the order of an integrator, the more precise the solution to the problem. However, higher-order time derivatives of particle coordinates should be stored.

- There exist symplectic integrators preserving the geometric structure of the Hamiltonian flow [88]. These integrators keep the fluctuations of energy bounded and allow for long-term energy conservation for simulations in the NVE ensemble (short-term energy conservation is unnecessary and sometimes results in a long-term energy drift).

- Some integrator schemes are time-reversible (as Newton's equations of motion). A general way to construct a reversible integrator uses the Trotter factorization of the Liouville propagator [77]. Numerical experiments show that symmetric (reversible) methods share similar properties with symplectic methods for certain systems [87].

- There exist single-step and multi-step methods. In a single-step method, the state of a system at the next step only depends on the previous step's state, while in a multi-step method the next state depends on the states at several previous steps.

The choice of the integration step has a crucial importance for the simulation method. In practice, the time step should be short compared to the vibration period of the particle. Usually, it is chosen to be about 10 times smaller than the shortest vibration period in the simulation. On the other hand, for the efficiency of the method, the time step should be chosen as large as possible.

There exist methods that increase the time step of the simulation [89, 90, 91, 92, 93, 94], allowing for faster simulations and larger time scales. This can be achieved, for example, by separating fast and slow degrees of freedom in the system [89] (in the general case [77]), by changing the mass tensor [90, 91, 94], by using an adaptive time step [92], etc. The internal coordinate method (ICMD) [139] is sometimes used in DNA modeling studies to use larger time steps.

The simulation time step may also be increased by constraining bonds in the system to remove high frequency modes. This idea is used in the SHAKE [30] and the RATTLE [31] methods (and their numerous variants), the penalty function method [95], and the methods modifying the potential [96].

1.3.3.2 Monte Carlo and efficient sampling

A large variety of techniques aimed at accelerating sampling has been developed [97, 85, 98].

For example, several modifications of the traditional Monte Carlo method exist: Gibbs ensemble Monte Carlo [99], configurational bias Monte Carlo [29] and others [100, 101, 102], as well as Monte Carlo methods for quantum computations [140]. For rare events, kinetic Monte Carlo may also be used [103]. Some algorithms such as hybrid Monte Carlo [84, 104] try to combine the advantages of MD and MC.

Some sampling methods use molecular dynamics simulations. For example, in the replica exchange approach [85] (the generalized-ensemble algorithm [141]) several non-interacting copies of the original system are simulated with different temperatures which are regularly swapped with some probability dependent on the energies of these copies. Statistical averages in this case are obtained from the system copies with the desired temperature. This method is widely used in protein folding.

In the LES method [98] a small subsystem of interest is cloned, resulting in several non-interacting copies, and the rest of the system feels an average potential from these copies. Phase space is therefore better sampled than in traditional MD.

Accelerated molecular dynamics [105, 97] allows to increase the time scale of simulations of rare events by modifying the potential (adding a term to it).

Steered molecular dynamics (SMD) [106] is a method that allows to observe biological processes at time scales accessible by molecular simulations. The general idea of this method is to apply external forces on one or more atoms to study the response of the system under various conditions. For example, a ligand-protein interaction may be studied by pulling the ligand out of the binding site.

The computations of a specific class of thermodynamical properties (those that cannot be obtained by simple averaging of some function of coordinates and momenta over all particles), *e.g.* the Helmholtz free energy and, more usefully, their differences between various states, have been extensively studied [107]. In particular, solvation free energy [19, 108, 109] has been of a high interest because the solvent often represents an indispensable part of a molecular system.

1.3.4 Hardware

We now discuss accelerating molecular simulations on a hardware.

If a problem may be easily subdivided into several smaller problems that can be solved simultaneously, parallel computing is a natural accelerating technique: calculations are split between several processing units and then gathered to provide the final result. One way to

perform molecular simulations in parallel is to separate computations inside a single simulation. This strategy may be applied to the methods as divide-and-conquer algorithms [21] or domain decomposition approach [110, 111]. Another option is to perform multiple simulations in parallel as for, *e.g*, replica exchange method [85].

Modern hardware provides many possibilities for parallel computing. For example, multicore Central Processing Units (CPUs) are now common even for desktop computers. Graphics Processing Units (GPUs) as well can accelerate N-body simulations [112, 113] or Monte Carlo simulations [114]. Heavy computations may be carried out on clusters or supercomputers [115]. There even exist supercomputers specially designed for molecular simulations [116, 117]. Distributed systems, that consist of a large number of autonomous computers interacting through a network, may help to solve important problems, such as protein folding [118].

1.3.5 Software

Many of the methods discussed above combined with other optimization techniques are integrated in a variety of sophisticated force fields and simulation packages that have been produced for both classical and quantum simulation, including for example CHARMM [119], AMBER [120], ABINIT [121], Desmond [122], NAMD [123], GROMOS [124] and GROMACS [125], LAMMPS, SIESTA, VASP, TINKER, YASARA, etc.

Some of these tools are open source, while others are available commercially, sometimes via integrating applications: Discovery Studio, Materials Studio, MedeA, MOE, Ascalaph Designer, BOSS, Spartan, Maestro, NanoEngineer-1, etc. Several tools are mostly concerned with visualization, but may sometimes be connected to simulation packages: VMD [2], PyMol [1], Zodiac, Avogadro, etc. The nanoHUB network [126] also includes a rich set of tools related to computational nanoscience.

To summarize, we illustrate molecular simulations on different time- and length-scales in Fig. 1.3.1, taken from the Lecture Notes to MolSim 2012.

1.4 Contributions

The main goal of this thesis was to contribute to accelerating molecular simulations. Precisely, in this work we considered algorithms that reduce the number of degrees of freedom in the system, while staying at particle resolution to compute the potential energy and forces. In these algorithms, some blocks of atoms are moving together as rigid bodies, which results in a smaller number of forces to be updated at each time step, since forces inside rigid bodies are constant and do not have to be updated. Since force computations represent an important part of the molecular simulation calculations, this approach may result in significant speedups.

The force evaluation process typically consists in two parts (we consider the potential to be pairwise for simplicity). First, neighbor search is performed: a neighbor list, *i.e.* a

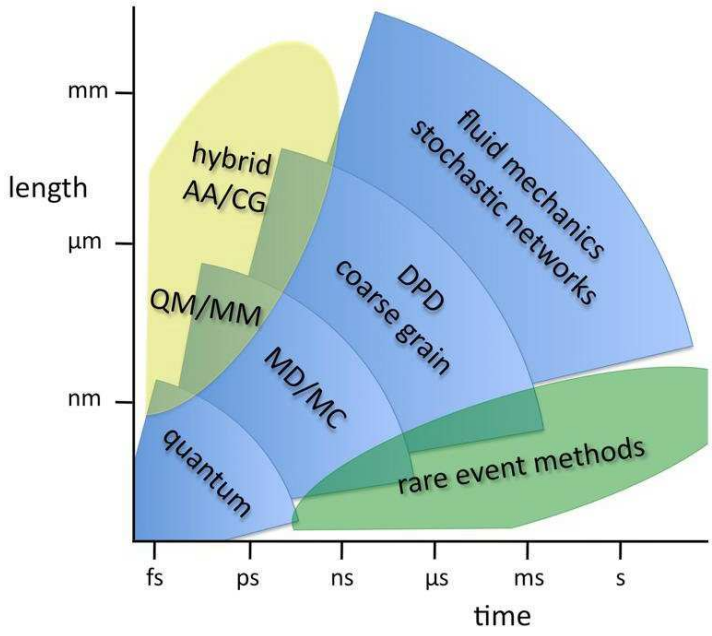


Figure 1.3.1: Molecular simulations on different time-length scales. Picture taken from the Lecture Notes to MolSim 2012.

list of all pairs of atoms that are closer than the predefined cutoff distance and are, thus, considered interacting, is computed. Second, the forces corresponding to the pairs of atoms in the neighbor list are calculated. It turns out that if large blocks of atoms are moving together, the neighbor search step may be accelerated. In the first part of this work, we present a study that compares different methods to perform neighbor search between large rigid groups of atoms. More precisely, we compare the traditional grid-based algorithm to a series of hierarchy-based algorithms that use bounding volumes to rapidly eliminate large groups of irrelevant pairs of atoms during the neighbor search. The latter algorithms have traditionally been used in computational geometry, with applications in computer graphics, robotics, virtual reality, etc. We compare the performance of these algorithms when varying several parameters: the size of the molecules, the average distance between them, the cutoff distance, as well as the type of bounding volume used in the culling hierarchy (AABB, OBB, wrapped or layered spheres). We demonstrate that for relatively large systems (>100 000 atoms) the algorithm based on the hierarchy of wrapped spheres shows the best results, and the traditional grid-based algorithm gives the worst timings. For small systems, however, the grid-based algorithm and the one based on the wrapped sphere hierarchy are beneficial. These results have been published in the Journal of Computational Chemistry [127].

Having noticed that the hierarchical approaches are often beneficial for simulations where large blocks of atoms might be considered rigid, we found out that no general method of constructing of such a hierarchy was proposed. That is why, in the second part of this thesis,

we introduce such an algorithm. Precisely, we present a fast general algorithm for the complete construction of a hierarchical representation of a molecular system. This three-step algorithm treats the input molecular system as a graph in which vertices represent atoms or pseudo-atoms, and edges represent covalent bonds. We demonstrate the performance of our algorithm on a set of specifically tailored difficult cases, as well as on a large subset of molecular graphs extracted from the Protein Data Bank [128]. Finally, we demonstrate an application of our hierarchy construction algorithm to adaptive torsion-angle molecular mechanics. These results have also been published in the Journal of Computational Chemistry [129].

The adaptive methods based on freezing some degrees of freedom as [78, 79, 41, 80] have never been supplemented by an analysis of the properties of the obtained simulations (for example the stability of such simulations). For applications to computer graphics, this was not important, since simulations have to be visually plausible but not necessarily theoretically rigorous. In molecular dynamics simulations, though, it is crucial to theoretically analyze every new approach and check whether it produces simulations that sample correctly the phase space.

That is why, in the third part of this thesis, we present a novel, mathematically-grounded method based on Hamiltonian dynamics, that switches positional degrees of freedom on and off, which reduces the number of forces to be updated at each time step, and may significantly speed up simulations. This method is suitable for a more general class of simulations: particle simulations, and we call it ARPS: Adaptively Restrained Particle Simulations. Our approach has numerous advantages. For example, it is able to produce stable, energy-preserving simulations, and, when performing constant-energy simulations, it allows users to finely and continuously trade between precision and computational cost, and rapidly obtain approximate trajectories. This trade-off between precision and cost may be chosen for each particle independently, so that users may arbitrarily focus ARPS on specific regions of the simulated system (*e.g.* a polymer in a solvent). Another important advantage is that, when performing Adaptively Restrained Molecular Dynamics (ARMD) in the canonical (NVT) ensemble, correct static equilibrium properties can be computed. We propose a general approach based on a modified inverse inertia matrix in the Hamiltonian, as well as a particular choice of this matrix to perform simulations in Cartesian coordinates and freeze and release particle positions independently. To perform efficient simulations, we propose incremental algorithms for the force update. We illustrate ARPS on several numerical experiments, including (a) a collision cascade example that demonstrates how ARPS make it possible to smoothly trade between precision and speed, and (b) a polymer-in-solvent study that shows how one may efficiently compute correct static equilibrium properties with ARPS. These results have been published in Physical Review Letters [130].

A natural way to proceed was, then, to combine a hierarchical approach to ARPS. In the last Chapter of the thesis we present such an approach. Precisely, we introduce another choice of the inverse inertia matrix and the resulting simulations, called hierarchical Adaptively Restrained Particle Simulations (hierarchical ARPS). In this case, the restraining function for each particle depends not only on the properties of this particle but also on the properties of

some other particles. Therefore, in this case, we switch on and off *relative* positional degrees of freedom in the system (*i.e.* group particles together into rigid bodies), and not simply the positional degrees of freedom of individual particles.

To summarize, in this thesis we propose several new general algorithms that speed up molecular simulations as well as compare existing methods to find the most suitable for particular simulations. We provide methods that are specifically designed for certain types of simulations as well as a general approach to perform particle simulations.

This dissertation is organized as follows. First, in Chapter 2 we present the comparative study of the methods performing neighbor search for large rigid molecules. Then, in Chapter 3 we present a novel algorithm aimed at constructing binary-tree representations of general molecular graphs. Later on, in Chapter 4 we describe a new approach to accelerating molecular (and, more generally, particle) simulations by switching partilce positional degrees of freedom on and off during the simulation (ARPS). Then, in Chapter 5 we combine a hierarchical approach to this novel method, which results in hierarchical ARPS. Finally, in Chapter 6 we provide conclusions and possible future work.

Chapter 2

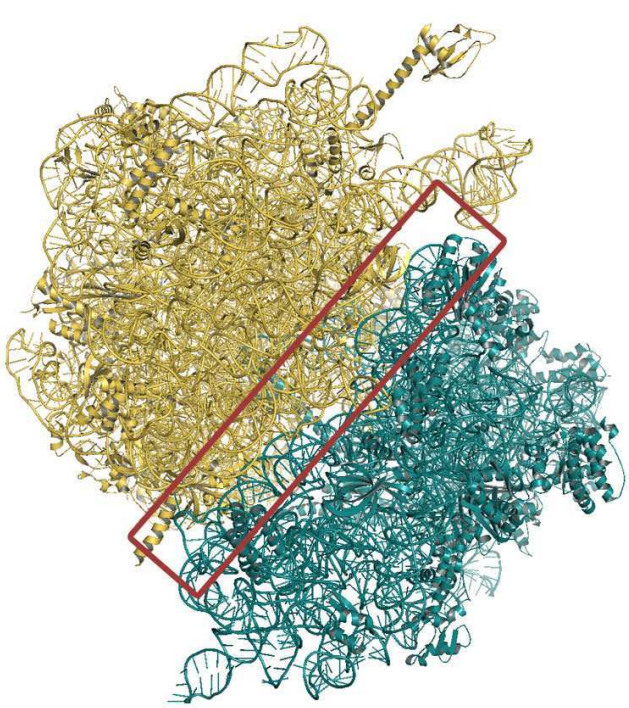
Comparison of neighbor search algorithms for large rigid molecules

Comme indiqué dans l'Introduction, les calculs de force représentent une part significative des calculs de simulation moléculaire. La détermination rapide des atomes voisins est une étape essentielle dans les calculs de force, et il existe de nombreux algorithmes pour calculer efficacement les listes de voisins. Cependant, la plupart de ces algorithmes sont généraux et ne sont pas spécifiquement conçus pour un type d'application donné. En conséquence, bien que leur performance moyenne soit satisfaisante, ils pourraient être inappropriés dans certains domaines d'application spécifiques. Dans ce Chapitre, nous étudions le cas de la détection des interactions entre les grandes molécules rigides, avec des applications, par exemple, pour le docking moléculaire des objets rigides, les simulations Monte Carlo de l'auto-assemblage ou de la diffusion moléculaire, ou encore la dynamique moléculaire des objets rigides.

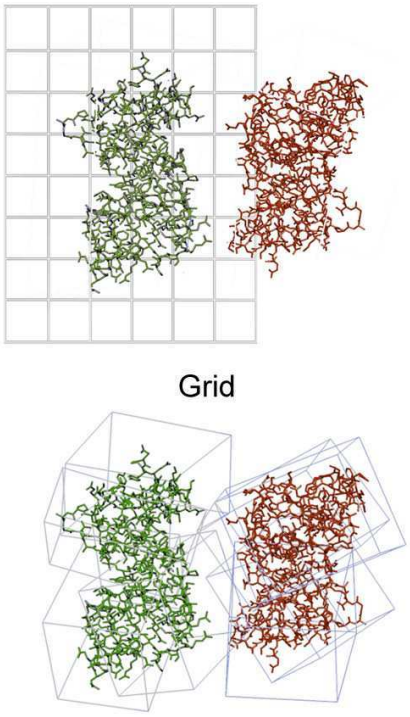
Plus précisément, nous comparons l'algorithme traditionnel utilisant une grille à une série d'algorithmes fondés sur la hiérarchie des volumes englobants, afin d'éliminer rapidement les grands groupes de paires d'atomes non pertinents lors de la recherche des voisins. Nous comparons les performances de ces algorithmes en variant plusieurs paramètres: la taille des molécules, la distance moyenne entre elles, la distance de coupure, ainsi que le type de volume englobant utilisé dans ladite hiérarchie (AABB, OBB, 2 types de sphères). Nous démontrons que pour des systèmes relativement grands (>$100\,000$ *atomes) l'algorithme fondé sur la hiérarchie des sphères montre les meilleurs résultats, et que l'algorithme traditionnel, utilisant la grille, donne les pires. Pour les petits systèmes, cependant, l'algorithme avec une grille et celui de la hiérarchie de sphères sont les plus efficaces.*

As discussed in the general Introduction, force computations represent a significant part of molecular simulation calculations. Fast determination of neighboring atoms is an essential step in force computations, and there exists a variety of algorithms to efficiently compute neighbor lists. However, most of these algorithms are general, and not specifically designed for a given type of application. As a result, although their average performance is satisfactory, they might be inappropriate in some specific application domains. In this Chapter, we study the case of detecting neighbors between large rigid molecules, which has applications in e.g. rigid body molecular docking, Monte Carlo simulations of molecular self-assembly or diffusion, and rigid body molecular dynamics simulations.

More precisely, we compare the traditional grid-based algorithm to a series of hierarchy-based algorithms that use bounding volumes to rapidly eliminate large groups of irrelevant pairs of atoms during the neighbor search. We compare the performance of these algorithms varying several parameters: the size of the molecules, the average distance between them, the cutoff distance, as well as the type of bounding volume used in the culling hierarchy (AABB, OBB, wrapped or layered spheres). We demonstrate that for relatively large systems (>100 000 *atoms) the algorithm based on the hierarchy of wrapped spheres shows the best results and the traditional grid-based algorithm gives the worst timings. For small systems, however, the grid-based algorithm and the one based on hierarchies of wrapped spheres are beneficial.*



Neighbor List Construction



Bounding Volumes

2.1 Introduction

As discussed in the previous Chapter, force computations represent a significant part of molecular simulation calculations. However, when large blocks of atoms are moving together as rigid bodies, the force update may be accelerated. In this Chapter, we present a comparative study on the methods designed for the first part of the force update procedure: neighbor search. We show how these computations may be accelerated in the case of two large rigid molecules.

The first step of the force updating process is to determine all interacting atoms in the system. Then, the corresponding interaction forces should be evaluated. Often, in potential functions, only pairwise interactions are taken into account and a *cutoff distance* is introduced, *i.e.* a maximum distance below which two atoms are considered to be interacting. Thus, to determine all interacting atoms, one needs to compute a *neighbor list*, that is to find all pairs of atoms that are closer than the predefined cutoff distance. This step may become the main cost for some systems. This is the case *e.g.* for systems consisting of two large rigid molecules with a small contact area, or when a small cutoff distance is used. For such systems, the time spent on evaluating forces can be significantly smaller than the neighbor list construction time.

Rapid construction of neighbor lists is a general problem that has been addressed in several domains, including robotics, computer graphics, computational geometry, etc. [23, 142]. In particular, it is a crucial step of the scoring phase of the computer-aided protein-protein docking simulations [5, 131, 132]. During this phase, a large number of generated docked conformations with favorable surface complementarity are ranked using a scoring function that approximates the free energy.

Several classes of neighbor search algorithms exist. Space-partitioning algorithms use a grid to detect collisions between two rigid bodies [143], deformable objects [144] or to simulate particle-based fluids [145, 146]. Algorithms relying on polytrees [147, 74, 148, 149] or R-trees [150] are also partially based on space partitioning. Object-partitioning algorithms use hierarchies of bounding volumes (BV-hierarchies) [151] and binary space partitioning trees (BSP-trees) [152]. Other algorithms have been specifically tailored for event-driven molecular dynamics simulations [153, 154], collision detection and self-collision computations for highly articulated macro-molecular systems [155, 156, 157, 158], and simulations of soft objects [159].

The performance of these algorithms may vary: some of them are slower to initialize, some need to update their underlying data structures during the simulation, etc. As a result, some algorithms may appear more suitable than others, depending on the application[1].

In molecular simulations, popular neighbor search algorithms are based on grids or Verlet lists [23, 161, 162, 163, 164, 165, 166, 167, 168]. However, to compute a list of interacting atoms in rigid body molecular docking [132, 169, 170], one can benefit from algorithms designed for collision detection between geometrical primitives [171, 172], which rely on hierarchies of bounding volumes.

[1] For example, van Gunsteren et al. [160] discuss the choice of the best algorithm for nonrectangular periodic systems.

In this Chapter, we compare preprocessing times and computational performances of five neighbor list construction algorithms. The first one is the grid-based method, and the four others use hierarchies of three popular types of bounding volumes: axis-aligned bounding volumes [173] (AABB), oriented bounding boxes [151, 172] (OBB) and spherical bounding volumes [174, 175] [2]. We perform the comparison for small biological complexes as well as systems containing as much as a few million atoms (*e.g.* two virus capsids), depending on several parameters (cutoff distance, intermolecular distance, etc.).

The rest of the Chapter is organized as follows. First, we describe all five algorithms to be compared. Then, we present experimental results for the biomolecular systems described above. Finally, we provide some conclusions on the performed comparison and discuss possible extensions.

2.2 Neighbor-search algorithms

In this section we describe the five algorithms we compare: the grid-based algorithm and the four algorithms relying on hierarchies of different types of bounding volumes. These algorithms aim at determining all pairs of atoms such that (a) one atom belongs to the first molecule and the other atom belongs to the second molecule, and (b) the distance between the two atoms is less than the cutoff value.

2.2.1 Grid-based algorithm

The idea behind the grid-based algorithm is a traditional cell-list approach [23].

As an input of this algorithm we consider two molecules that we call receptor and ligand (a 2D example is shown in Fig. 2.2.1, top figure).

To initialize this method, we first find the smallest axis-aligned parallelepiped enclosing one of the molecules (usually the bigger one: the receptor), and enlarge it by the cutoff distance in all directions of the outward normals to its faces. Then, this parallelepiped is divided into cubic cells with an edge size equal to the cutoff distance[3]. Finally, the receptor atoms are distributed to the grid cells (Fig. 2.2.1 A, bottom).

To compute a neighbor list, each atom of the other molecule (ligand) is examined. First, we check whether this ligand atom may be mapped to some cell of the constructed grid or it lies too far from the receptor molecule. If the ligand atom belongs to some grid cell, it is checked for proximity with the receptor atoms in the current cell and those in the grid-cells adjacent to the current cell (Fig. 2.2.1 B, bottom). Since the cell-size is equal to the cutoff distance, exploring this set of cells is sufficient to find all receptor atoms interacting with the current ligand atom.

[2] This can be extended to the approach using discretely oriented polytopes defined by k vectors (k-DOPs) [171].

[3] It is not always possible to divide the parallelepiped into cubic cells with the predefined edge size. To solve this problem, one can either slightly enlarge the parallelepiped in the necessary directions, or consider some boundary cells as non-cubic.

Algorithm 2.1 Algorithm for computing a neighbor list for two rigid bodies with a certain cutoff distance using a grid.

PARAMETERS: rigid body transform T (translation and rotation), cutoff distance.

```
for (all atoms a_i in the ligand molecule)
    if (the transformed atom T(a_i) can be mapped to some grid cell C)
        for (all test cells: cell C and its neighbors)
            for (all atoms b_j inside a test cell)
                if (distance between b_j and T(a_i) is less than cutoff)
                    add this pair of atoms to neighbor list
```

Due to the enlargement of the grid by the cutoff distance (by one-cell layer) in all axes directions, every ligand atom that is closer than the cutoff to any receptor atom will be mapped to some grid cell and checked for proximity with receptor atoms. Thus, the neighbor list constructed with this method will be complete[4].

Algorithm 2.1 describes the grid method in pseudo-code. It might be interesting for some applications (*e.g.* conformational space search for protein docking) to perform neighbor search between the receptor molecule and several different rigid body transformations (translation and rotation) of the ligand molecule. Therefore, the first input parameter for the algorithm is a rigid body transform. The second parameter is a cutoff distance.

2.2.2 Hierarchy-based algorithms

The general algorithm described in this section relies on hierarchical representations of molecular systems: each of the two molecules is associated to a binary tree. In both trees, each leaf node refers to a non-empty set of atoms, such that each atom in the molecule belongs to exactly one leaf node, while each internal node represents the system that is obtained by joining together the non-intersecting subsystems referred to by its children. Finally, each node in the tree is associated to a bounding volume, *i.e.* a geometrical primitive which encloses all atoms corresponding to the node.

An example of a binary hierarchy corresponding to a simple molecule is shown in Fig. 2.2.2.

2.2.2.1 Construction of the binary hierarchy

A binary decomposition of a molecule can be constructed in either a top-down or a bottom-up manner. Top-down algorithms start from the whole set of atoms and split the system

[4]Note that we have not tested the grid-based method with other cell sizes, because we believe it would not significantly speed up the results. Indeed, considering a grid with a larger cell size will result in more unnecessary proximity queries between atoms ("false positives"). A smaller cell size, on the other hand, will require exploring not only cells that are adjacent to the current cell, but also some that are farther apart. This will reduce the number of false positives, but will lead to a larger number of memory accesses to the grid structure, which is also costly.

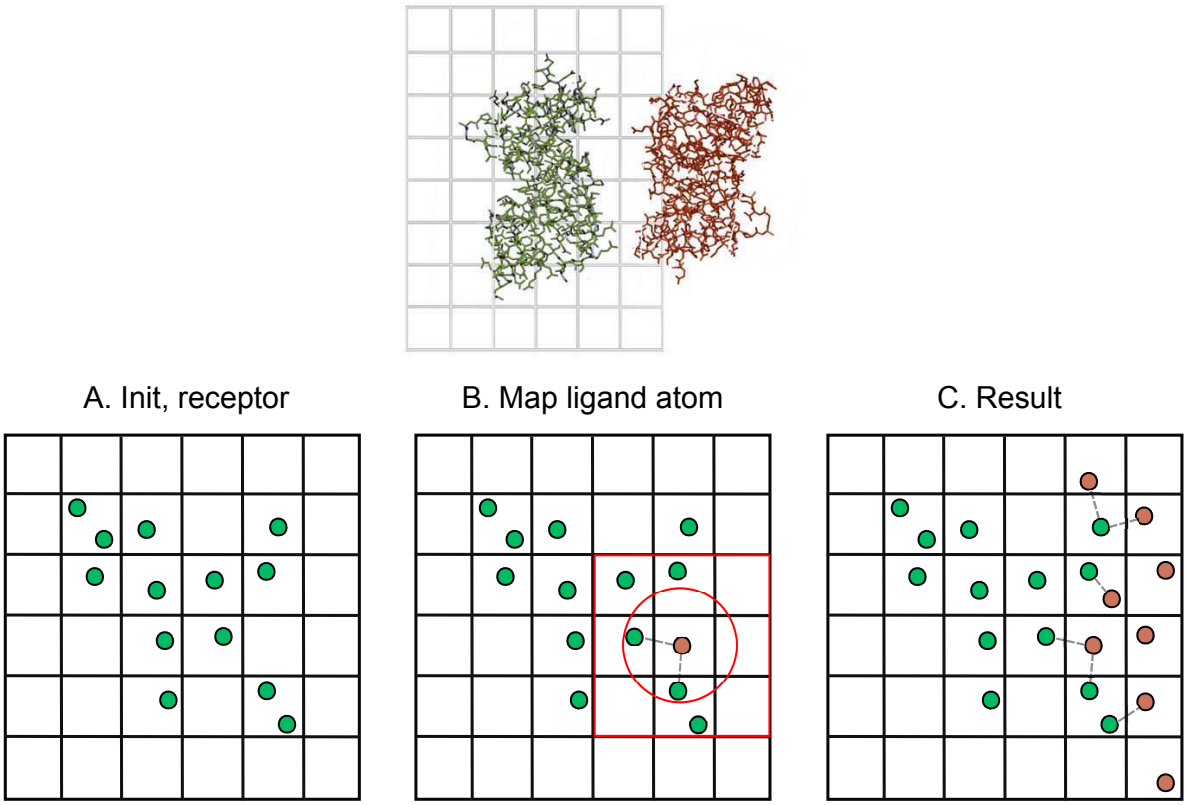


Figure 2.2.1: Grid-based algorithm for neighbor list construction in 2D. **Top figure**: method overview. The receptor molecule is green (left molecule), the ligand molecule is brown (right molecule). **Bottom figure**: the three main steps of the grid-based neighbor search algorithm. **A**. Method initialization: receptor atoms (green or light grey circles) are distributed to the grid. **B**. Each atom of the ligand molecule (brown or dark grey circle) is individually mapped to the grid. Receptor atoms interacting with this ligand atom may only be situated in the cells adjacent to the cell of the ligand atom (these cells to be examined are enclosed into the red square). Receptor atoms interacting with the ligand atom are situated within the cutoff distance circle (red circle). Pairs of atoms to be included into the neighbor list are marked with grey dashed lines. **C**. Result of the algorithm's execution: all ligand atoms are mapped to the grid. All possible pairs to be included into the neighbor list are marked with grey dashed lines.

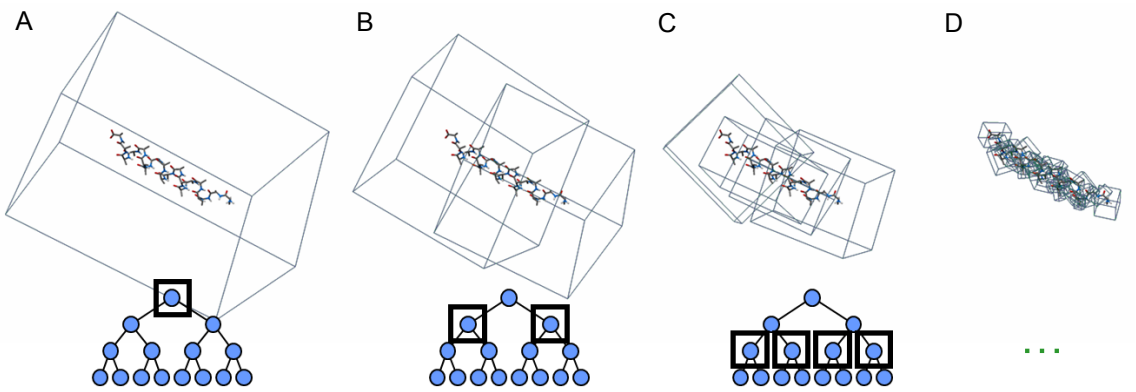


Figure 2.2.2: Hierarchical representation of a molecule. A. The whole molecule and the bounding volume enclosing it correspond to the root node of the tree. B. The two halves of the initial molecule and the two boxes enclosing these halves correspond to the child nodes of the root node. C. Halves of the halves of the initial molecule and their enclosing boxes correspond to internal nodes of the tree. For visual clarity bounding boxes are not tight.

recursively into two subsystems according to some rule [172, 176]. Bottom-up algorithms, on the contrary, group subsystems into one tree node starting from the individual atoms or groups of atoms considered as leaves [177, 178].

As will be seen from the algorithms description below, hierarchy-based algorithms benefit from a balanced tree, where the child nodes of each internal node are well-separated in space. This kind of trees is usually produced by top-down algorithms [172]. Although these algorithms result in $O(n \log n)$ time-complexity for a system of n atoms [172] (and the bottom-up approaches can be linear in running time[5]), they are still preferable, because in rigid body simulations, most hierarchy-based neighbor search algorithms are initialized only once, but are launched several times.

Hence, we choose the following procedure of the tree construction: we recursively split the system into two subsystems with an approximately equal number of particles inside, until the number of atoms in the system is not larger than a pre-determined, user-defined threshold m (main loop). We compute bounding volumes for the tree nodes at the same time.

Let us describe the splitting procedure in more detail.

At each iteration of the main loop, we divide a system into two subsystems by projecting atoms along a direction chosen from a predefined set of *candidate vectors*[6]. To choose this direction, we perform the same procedure for each candidate vector from the set. First, we sort atoms according to their coordinates along the vector. The first $n/2$ ordered atoms are assigned to the first subsystem and all the rest to the second subsystem. Then, we compute

[5] The simplest procedure would be pairwise assembling into the tree regardless of nodes properties.

[6] In our implementation, in this set we consider x, y and z axes, three eigenvectors of the covariance matrix for the system, and eight other vectors that are obtained as combinations of extreme values of the atom's coordinates along these first six vectors.

the tightest bounding volume enclosing each subsystem as follows:

- For axis-aligned bounding volumes we use minimal and maximal values of the atoms coordinates along x, y and z axes to obtain the vertices of the box.
- For oriented bounding volumes we use the covariance matrix of the subsystem [151]: axes of a new bounding volume are the eigenvectors of the covariance matrix for atoms coordinates[7] .
- For spherical bounding volumes we use the minimal enclosing sphere algorithm described in Ref. [180] (and references therein): a pivoting scheme resembling the simplex method for linear programming.

Finally, the direction that we choose to split the system is the candidate vector which results in the smallest sum of the volumes of the two subsystems bounding volumes. These bounding volumes are assigned to the child nodes of the current node (they become a part of the hierarchy of bounding volumes). The bounding volume of the root node is computed as a tightest bounding volume enclosing the whole set of atoms.

Please note that in our implementation, we used not the real volumes of the BV's as geometrical objects, but some functions that we will call volumes later in this Chapter: for AABB's we considered as volumes their 3D diagonals, for spheres – their radii, and for OBB's – sums of the squares of the projection lengths of their sides to the coordinate axes. However, any other characteristic of the box may be used including its real volume.

Note also that, as a result, the topology of the binary tree may be different for each bounding volume type, since the splitting will depend on the bounding volume choice. This helps to produce bounding-volume hierarchies with good properties (balance and separation) for any type of bounding volume.

Using the procedure described above we can obtain three types of hierarchies of bounding volumes: AABB hierarchies, OBB hierarchies and so-called wrapped sphere hierarchy (WS hierarchies). Fig. 2.2.3 shows planar examples of two levels of AABB and OBB hierarchies. To construct layered sphere (LS) hierarchies [181]), however, we use the same tree topology as for WS hierarchies, but modify the spheres of internal nodes. Traversing the tree bottom-up, we compute for each internal node the smallest sphere bounding the spheres corresponding to its children (and not the atoms inside the spheres). Fig. 2.2.4 shows a planar example illustrating the difference between these two types of sphere hierarchies[8].

2.2.2.2 Neighbor search algorithm

The hierarchy-based algorithm is the same for all types of bounding volumes, and consists in a simultaneous traversal of the bounding-volume hierarchies corresponding to the two rigid

[7] Though, the method based on the covariance matrix is the most used, there exist other ways to compute an OBB for a system, *e.g.* the one described in Ref. [179].

[8] In Ref. [156] it is shown that, in the worst case, for a given set of n points, a bounding sphere in the layered hierarchy is at most a factor of $\sqrt{\log n}$ larger than the corresponding one in the wrapped hierarchy, and this bound is tight.

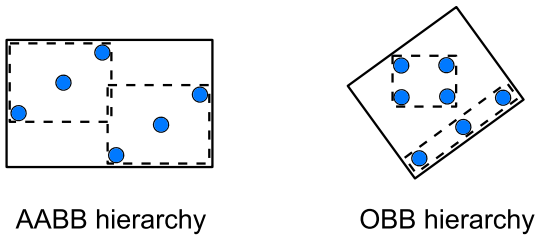


Figure 2.2.3: Two levels of the binary BV-hierarchy in 2D: for AABB's (left) and OBB's (right). BV's of a higher level of the hierarchy are shown with solid lines, child BV's of a lower level are represented by dashed lines.

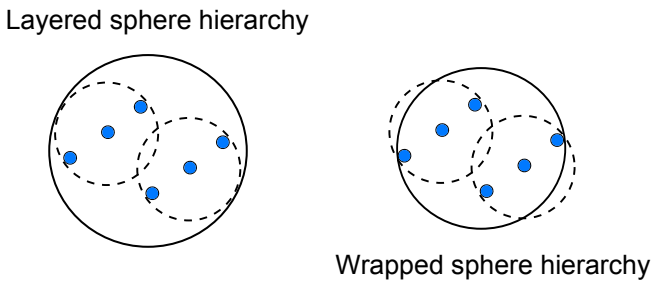


Figure 2.2.4: Two levels of the binary spherical BV-hierarchy in 2D: two child spheres are bounded (layered hierarchy) or the tightest bounding circle for the whole system is computed (wrapped hierarchy). BV's of a higher level of the hierarchy are shown with solid lines, child BV's of a lower level are represented by dashed lines.

molecules in a top-down fashion. The simultaneous traversal rapidly determines the pairs of interacting leaf bounding volumes. When two leaf bounding volumes are found to be interacting, pairwise distance queries between the atoms that they contain determine the pairs of atoms that are closer than the cutoff distance.

Let us now describe the hierarchy-based algorithm in more detail. This algorithm internally uses a *stack of pairs of bounding volumes* that is initialized with the pair of *root bounding volumes* of the BV-hierarchies. While this stack is not empty, we pop a pair of BV's out of the stack and perform a *proximity query* on this pair, *i.e.* compute the distance between the two bounding volumes. If this distance is larger than the cutoff value, then there can be no pairs of interacting atoms in the corresponding sub-trees, and we continue to the next iteration of the loop. If the bounding volumes are sufficiently close to each other (*i.e.* the distance between them is smaller than the cutoff distance), we refine the search as follows. If both BV's in the pair are hierarchy leaves, we compute distances between all pairs of atoms corresponding to them (performing no more than m^2 comparisons). Pairs of atoms closer than the cutoff distance are added to the neighbor list. If only one of the BV's in the pair is a leaf, we put to the stack two new pairs: the pair composed of the leaf BV and the left child of the non-leaf BV, and the pair composed of the leaf BV and the right child of a non-leaf BV. If both BV's are not leaves, we compare their volumes. Then, we put to the stack two pairs: the smaller BV (the one with a smaller volume) with the left and the right child of the larger BV (the one with a larger volume).

Algorithm 2.2 describes this method in pseudo-code. As for a grid-based method, two input parameters must be provided: a rigid body transform for a ligand molecule and a cutoff distance.

An example of a workflow of this algorithm with a zero cutoff (for visual clarity) for a hierarchy of OBB's is shown in Fig. 2.2.5.

2.2.2.3 Proximity queries for bounding volumes

The proximity query for a pair of bounding volumes mentioned in the algorithm determines whether the distance between them is less than the cutoff. It is specific for each type of bounding volumes.

- For two axis-aligned boxes this query performs as follows: both boxes are enlarged by a half of the cutoff distance, then, all intervals describing these boxes are checked for intersection. To optimize the calculation we enlarge all the AABB's in receptor and ligand by a half of the cutoff already while constructing the hierarchy.

- For two oriented bounding volumes this query is a slightly modified overlap test based on the separating axis theorem and described in Ref. [151]. In the original theorem we check along fifteen vectors (a sufficient set of axis tests) whether the sum of projections of the boxes half-sizes exceeds the projection of the vector-difference between centers of the boxes. In our case the first sum is compared to the vector-difference projection norm plus the cutoff distance. This modification is not equivalent to the obvious enlarging of

Algorithm 2.2 Algorithm for computing a neighbor list for two rigid bodies within a certain cutoff distance using a corresponding hierarchy of axis-aligned, oriented or spherical bounding volumes.

PARAMETERS: rigid body transform T (translation and rotation), cutoff distance.

clear stack S of pairs of BV's
push a pair to S: (root BV of the receptor; root BV of the ligand)

while (stack S is not empty)
    pop a pair of BV's from stack S: (A; B)
    if (distance between A and $T(B)$ is less than cutoff)
        if (A **and** B are leaf nodes of BV-hierarchies)
            for (all atoms a_i in A and all atoms b_j in B)
                if (distance between a_i and $T(b_j)$ is less than cutoff)
                    add this pair of atoms to neighbor list
        if (A is a leaf node **and** B is not a leaf node)
            push pairs (A; left child of B) and (A; right child of B) to stack S
        if (A is a not a leaf node **and** B is a leaf node)
            push pairs (left child of A; B) and (right child of A; B) to stack S
        if (neither A nor B **is** a leaf node)
            if (volume of A is greater than volume of B)
                push pairs (left child of A; B) and (right child of A; B) to stack S
            else
                push pairs(A; left child of B) and (A; right child of B) to stack S

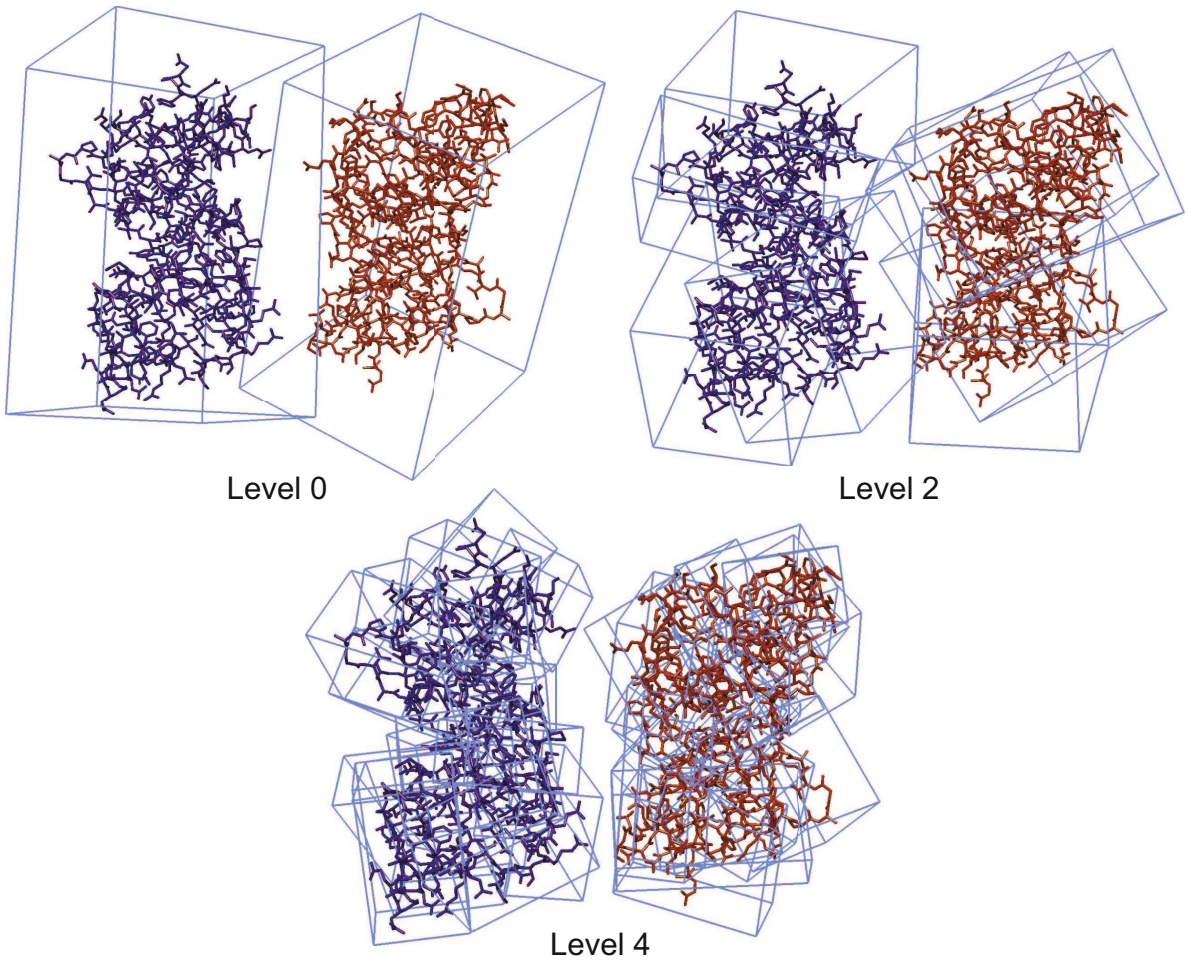


Figure 2.2.5: Neighbor search for two rigid bodies using a hierarchy of oriented bounding volumes (zero cutoff). Algorithm stops when bounding volumes corresponding to different rigid bodies do not overlap (level 4 on the picture), otherwise it goes down the hierarchy.

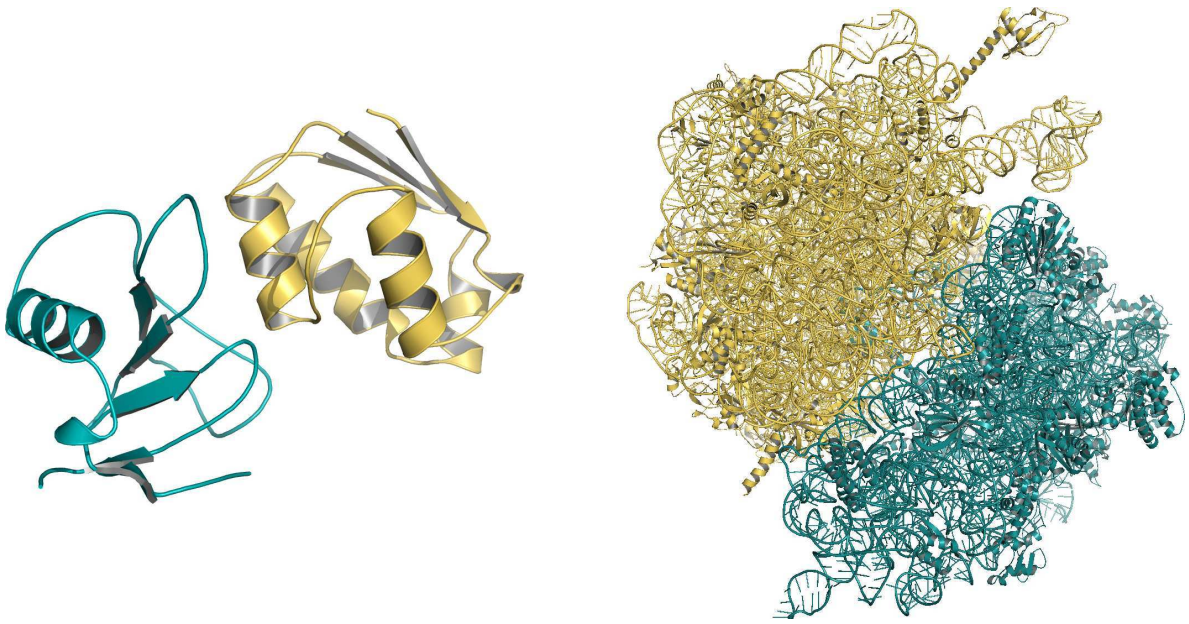


Figure 2.3.1: Ribonuclease Sa complex with barstar (PDB code 1AY7, chain A is green, chain B is yellow) and 70s ribosome (PDB codes 3MS1, yellow, and 3MR8, green). Images obtained with PyMol [1].

one of the boxes by a cutoff along each axis and applying the original separating axis theorem. In the latter case we would have unnecessarily enlarged (by the square root of the cutoff) the box in the directions of the vectors connecting the center of the box with its vertices. That would result in unnecessary pairs of interacting boxes, and, therefore, in an increased number of algorithm's operations.

- Two spherical bounding volumes are closer than the cutoff distance if the distance between their centers is less than a sum of their radii and the cutoff distance.

2.3 Results

In this section, we compare preprocessing times and computational performances of all algorithms described above: the grid method and the four methods based on hierarchies of bounding volumes. More precisely, we compare algorithms performances on two categories of benchmarks.

The first category of benchmark systems is a pair of biological complexes: a ribonuclease Sa complex with barstar (PDB code 1AY7, 751 and 737 atoms without water molecules, respectively), and a 70s ribosome (PDB codes 3MR8 and 3MS1, 56 047 and 91 063 atoms respectively). Both complexes are shown in Fig. 2.3.1.

Name	PDB code, first molecule	PDB code, second molecule	Number of atoms, first molecule	Number of atoms, second molecule	Autopsf tool used
Ribonuclease Sa complex with barstar	1AY7, chain A	1AY7, chain B	751	737	no
70s ribosome	3MS1	3MR8	91 063	56 047	no
Bluetongue virus capsids	2BTV	2BTV	3 588 900	3 588 900	yes
Apoferritins	1AEW	1AEW	39 672	39 672	yes

Table 2.1: Comparing neighbor search algorithms for large rigid molecules. Summary of the information on the benchmark systems.

The second category consists in two replicas of the *same* molecule: (1) a large virus capsid (bluetongue virus, asymmetric unit PDB code 2BTV, Fig. 2.3.2 top left, 3 588 900 atoms in the complete capsid, Fig. 2.3.2 top right), and (2) an apoferritin molecule (asymmetric unit PDB code 1AEW, 39 672 atoms in the complete molecule, Fig. 2.3.2 bottom). For these molecules we added missing hydrogens with the Automatic PSF Generation Plugin for the VMD software [2](using CHARMM19 force-filed).

We chose these molecular systems for benchmarks to compare the algorithms performances on well-studied (both experimentally and computationally) biological complexes of different size.

Information about the benchmark systems is summarized in Table 2.1.

For all systems described above we ran two types of tests: a *cutoff test* and a *distance test* introduced further.

All algorithms were implemented in C++ and (unless otherwise specified) run on an Intel Xeon X5450 3GHz processor with 16 GB of RAM, on a Windows Vista 64-bit operating system.

2.3.1 Basic operation time

Before comparing the overall performance of the five algorithms, we present a comparison of the times required to perform basic operations in hierarchy-based algorithms in our implementation. As the timings of interest are very small, they were averaged over 1 000 000 of identical tests.

The results for a BV-proximity query (*i.e.* a query to determine whether the distance between two bounding volumes is smaller than the cutoff distance) with translation and rotation of one of the bounding volumes are shown in Table 2.2.

The cost of the basic operation for the grid method (insertion of the atom to the grid) depends on the size of the grid cell (and, therefore, on the cutoff distance), and on the number

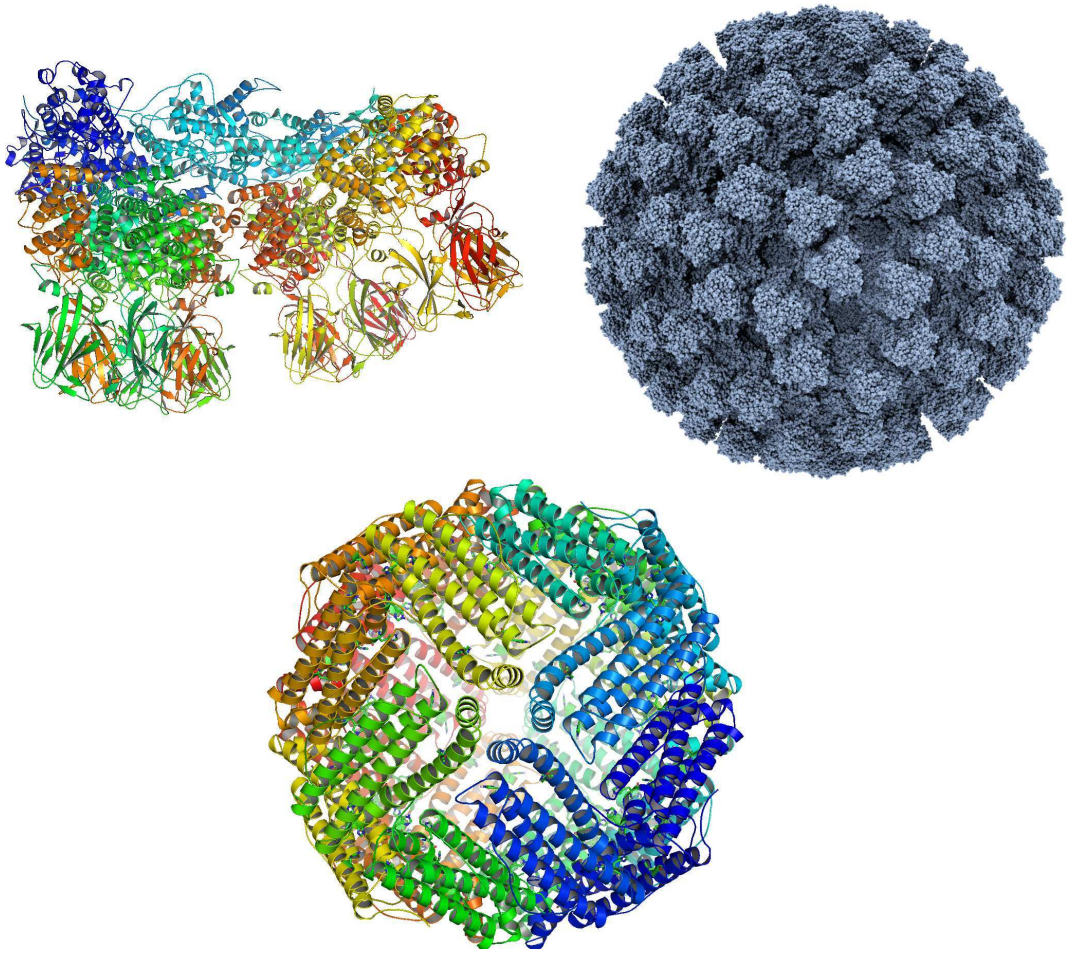


Figure 2.3.2: Top: Bluetongue virus capsid asymmetric unit (PDB code 2BTV) and the complete virus capsid. Bottom: apoferritin molecule (asymmetric unit code PDB code 1AEW). Images obtained with PyMol [1] and VMD [2].

Type of the BV	BV-proximity query (s)
Axis-aligned BV	1.275×10^{-7}
Oriented BV	1.610×10^{-7}
Spherical BV	2.643×10^{-8}

Table 2.2: Timings necessary for hierarchy-based algorithms to perform a basic operation: compare a distance between two bounding volumes with a cutoff value.

of atoms in each grid cell (and, therefore, on the density of the system).

Finally, in our implementation, the time to check whether the distance between two atoms is smaller than the cutoff distance is about 1.8×10^{-8} seconds.

2.3.2 Preprocessing time

We first compare preprocessing times for all described algorithms. As indicated above, initialization of the grid-based algorithm consists in creating the grid and distributing atoms of the receptor to the cells of this grid, while initialization of hierarchy-based algorithms consists in constructing two hierarchies of bounding volumes.

We perform this test for an apoferritin molecule. We present the results only for one molecule for the following reason. The so-called n-th element algorithm from the C++ Standard Template Library [182] (used to order the atoms according to their positions along some axis) results in a $O(n \log n)$ time complexity for the tree construction, where n is the number of atoms in the system. On the other hand, the running time of a grid construction algorithm for a fixed cutoff depends linearly on n. Hence, for bigger systems the difference between the timings for grid method and tree-based algorithms can only increase. Also, here we construct only one BV-hierarchy, which is the case for the benchmarks with two identical molecules.

We measured preprocessing times for the five algorithms. We varied values of the upper bound m on the number of atoms in the leaf node of the bounding-volume hierarchy (from 1 to 49 with a step of 3) and the cutoff distance for the grid algorithm (from 4 to 12.5 Angstroms (Å) with a step of 0.25 Å). Here and further on we use cutoff distances from the range of 4 to 14 Å. We choose these values for several reasons. First, the standard cutoff distance values in molecular dynamics simulations of biomolecules are about 12 Å. Second, knowledge-based scoring functions for molecular docking typically use cutoff distances between 6 and 12 Å [183].

The dependence of the grid algorithm's preprocessing time on the value of the cutoff distance is shown in Fig. 2.3.3 (left). We also plot the number of occupied cells in the grid constructed with the specified cutoff distance, as only these cells were created. The time decreases when the cutoff distance increases, because we have to create less grid cells. Average number of atoms in occupied grid cells varies from 3.75 to 77 and scales cubically in cutoff distance (Fig. 2.3.3, right).

The construction time for hierarchy-based algorithms depends on leaf size parameter m as shown in Fig. 2.3.4. Similarly, the time decreases when this parameter increases, since for larger values of m we perform less splittings of the system, which results in a smaller tree depth. Note that the latter dependency is not regular (there are "steps" on the plot) as we build the tree in a top-down manner, and it is not always possible to split the system into blocks of size m (thus, in each leaf node the number of atoms is smaller than or equal to m). As a result, the average number of atoms in the tree leaves remains the same for a range of cutoff distances. To illustrate this, we plotted the true average number of atoms in the leaf bounding volumes on the same figure. It can be clearly seen that construction times are directly related to the average number of atoms in leaf nodes.

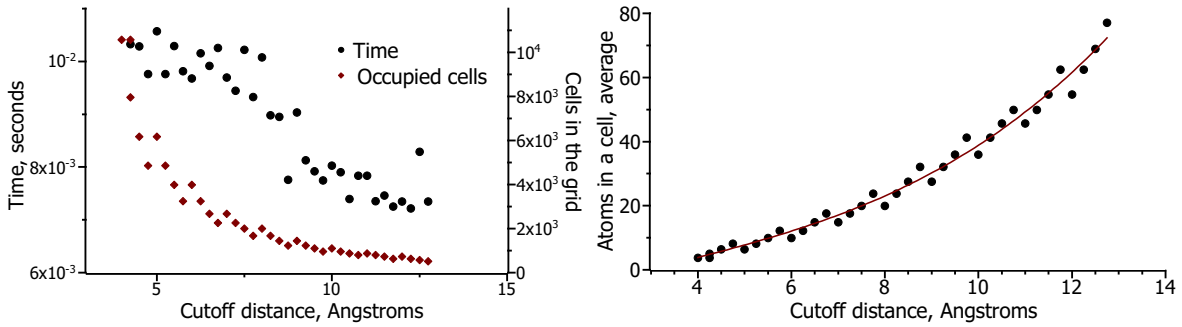


Figure 2.3.3: Left plot: preprocessing time of the grid algorithm as a function of the cutoff distance (left axis), and the number of the cells in the constructed grid as a function of the cutoff distance (right axis). Right plot: average number of atoms in occupied grid cells, solid line represents a cubic fit.

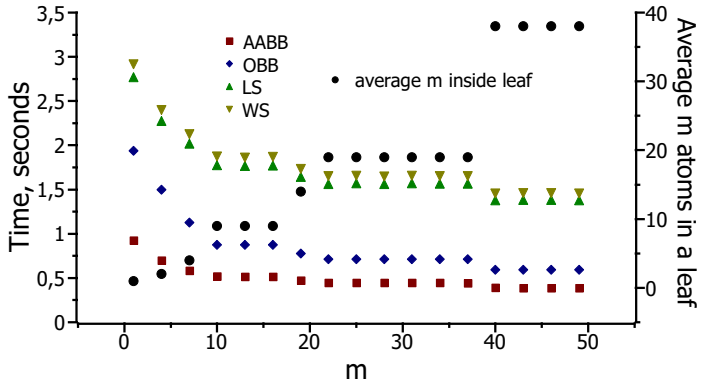


Figure 2.3.4: Preprocessing times of all hierarchy-based algorithms as functions of m (left axis). Corresponding true average number of atoms in the leaf BV of the hierarchy (right axis).

Overall, for all combinations of values of the cutoff and leaf size parameter m indicated above, the preprocessing time for the grid method is always less than preprocessing times of all hierarchy-based algorithms.

It should be emphasized, however, that bounding-volume hierarchies associated to rigid molecules have to be computed *only once*, and may be re-used when the relative position between the two rigid molecules is changed. Therefore, preprocessing time is important, but not crucial, and we now compare the *query time* of all described neighbor-search algorithms on two types of tests.

2.3.3 Cutoff test

For the first test type (cutoff test) we do not change the distance between the molecules in the system of interest but vary the cutoff distance (and, therefore, the number of the pairs in the neighbor list).

2.3.3.1 Choice of leaf size parameter m

The running-time complexity of the hierarchy-based neighbor search algorithm depends on the leaf size parameter m. Therefore, before running cutoff tests as described above for all benchmark systems, we perform a test to numerically determine optimal values of this parameter for different types of bounding-volume hierarchies. A more detailed analysis on the choice of m is presented in the Appendix to this Chapter. We run a cutoff test for the two molecules comprising the 70s ribosome, where the cutoff value was varied from 4 to 12 Å with a step of 4 Å, without altering the position of either molecule. The values of the leaf size parameter m were also varied from 1 to 46 with a step of 3.

Figure 2.3.5 (left), shows the dependency of AABB- and OBB-based algorithms on m. For these methods, it appears that the smaller timings correspond to values of m between 8 and 12. Figure 2.3.5 (right), shows the dependency of LS- and WS-based algorithms on m. In this case, optimal values of m appear to be between 4 and 7. As a result, further on we use the following values for m: 8 for hierarchies of AABB's and OBB's, and 4 for hierarchies of spheres.

2.3.3.2 Biological complexes

We performed the cutoff test for the Sa-barstar complex and the 70s ribosome. We preserved experimentally resolved coordinates of the molecules, fixed the leaf size parameter $m = 8$ for AABB and OBB hierarchies and $m = 4$ for both sphere hierarchies, and varied the cutoff distance from 4 to 13.5 Å with a step of 0.5 Å. Results are shown in Fig. 2.3.6 on the left and right plots, respectively. Dependencies of the total times for all algorithms on the number of the pairs in the neighbor list N are well-fitted by the following functions: $c_i N^{a_i} + b_i$ ($a_i < 1$, $i = 1..5$). There is a non-zero coefficient b_i, suggesting that even for zero pairs of atoms in contact, the time of the algorithm execution will be positive. This happens because we first need to reach a certain level down the BV-hierarchy to determine

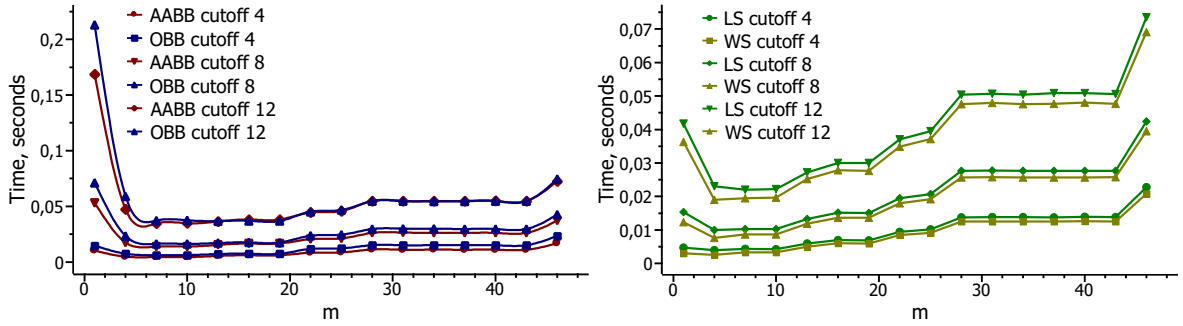


Figure 2.3.5: Cutoff test. Time dependence of AABB- and OBB-based algorithms (left) and the two algorithms based on spheres (right) on the leaf size parameter m. Cutoff distances expressed in Angstroms.

that there are no more bounding volumes within the cutoff distance and stop the algorithm. This stopping level and, therefore, the coefficient b_i will depend on the conformations of the molecules, the distance between them and the cutoff value. Coefficient c_i represents the costs of elementary operations. Hence, these two parameters are different for different types of bounding volumes. A more detailed complexity analysis for the neighbor list construction algorithm using a AABB-hierarchy can be found in Ref. [76].

For the grid algorithm, the size of each grid cell and, therefore, the average number of atoms in a cell, depends on the cutoff distance. Of course, the number of occupied cells in the grid depends also on the shape of the molecule. Thus, the number of comparison operations, and hence the total time, depends on the shape of the molecule. As can be seen on the plots (Fig. 2.3.6), for a relatively small complex, the grid algorithm performs better, but the timings for the WS-based algorithm are very similar. For a larger molecule such as the ribosome, however, the WS-based algorithm becomes beneficial. In this case, indeed, the ratio of the contact area to the size of the molecules is smaller. Thus, pairs of large distant subsystems are rapidly eliminated by the hierarchy-based algorithm, and the detailed neighbor search is concentrated in a smaller area. For both complexes, among all hierarchy-based algorithms, the one relying on the hierarchy of wrapped spheres has the best timings.

In some cases, the neighbor list construction time can be significantly more important than the force evaluation time. For large rigid systems, the number of force evaluations is roughly proportional to the contact area between the two molecules, whereas the complexity of neighbor search using a grid is linear in the number of atoms. To experimentally compare computational costs, we used a simple potential energy function: the Lennard-Jones potential, for which a single force evaluation in our implementation takes about 8.3×10^{-8} seconds.

For the two molecules comprising the 70s ribosome, there are 1 174 pairs of atoms in contact

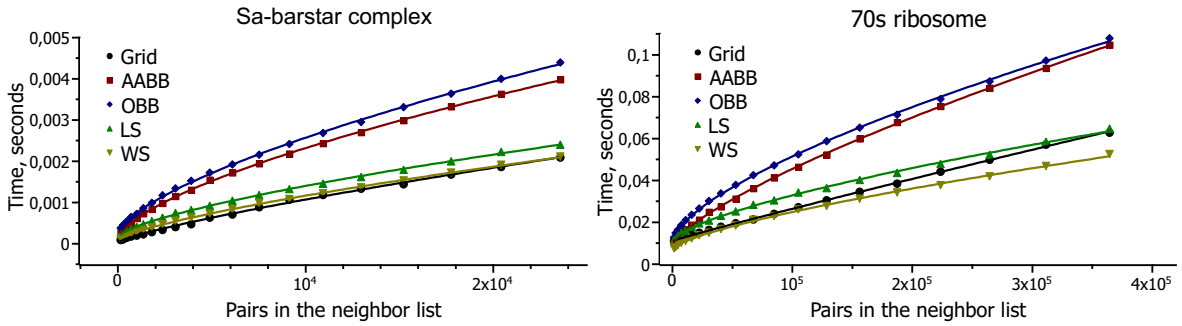


Figure 2.3.6: Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact for Sa-barstar complex (left) and 70s ribosome (right). Solid lines represent fitting curves.

within the cutoff distance 4 Å. Therefore, the force evaluation time is $1\,174 \times 8.3 \times 10^{-8} \approx 1 \times 10^{-4}$ seconds. The time to construct the neighbor list with a grid method is 0.011 seconds, whereas with a hierarchy of WS's this time is 0.008 seconds. Thus, neighbor search construction is about two orders of magnitude slower than force evaluation.

2.3.3.3 Two bluetongue virus capsids

We performed the cutoff test for two bluetongue virus capsids, one of them being shifted by 689 Å along the x-axis: an approximate diameter of the capsid's enclosing sphere. This shift eliminates steric clashes between the two molecules and results in a wide range of neighbor list sizes as cutoff varies. We chose $m = 8$ (for AABB and OBB hierarchies) and $m = 4$ (for LS and WS hierarchies), and varied the cutoff from 4 to 12 Å with a step 0.5 Å. The obtained timings are displayed in Fig. 2.3.7 as functions of the number of pairs in contact. We plot on the left, in the logarithmic scale, the results for all algorithms, and on the right, in linear scale, the timings for hierarchy-based algorithms only. Again, functions of the form $c_i N^{a_i} + b_i$ ($a_i < 1,\ i = 1..5$) fit well all obtained timings. In this case, for two large rigid molecules with a relatively small area of contact (a realistic situation for such large molecules), hierarchical methods clearly outperform the grid method. Again, among all hierarchy-based algorithms the one based on wrapped spheres shows the best results.

2.3.3.4 Two apoferritin molecules

Finally, we ran this test for a system of two apoferritin molecules, one of them shifted by 122 Å along the x-axis. This shift is an approximate diameter of the apoferritin's enclosing sphere. Again, the chosen shift eliminates steric clashes between the two molecules, and results in a

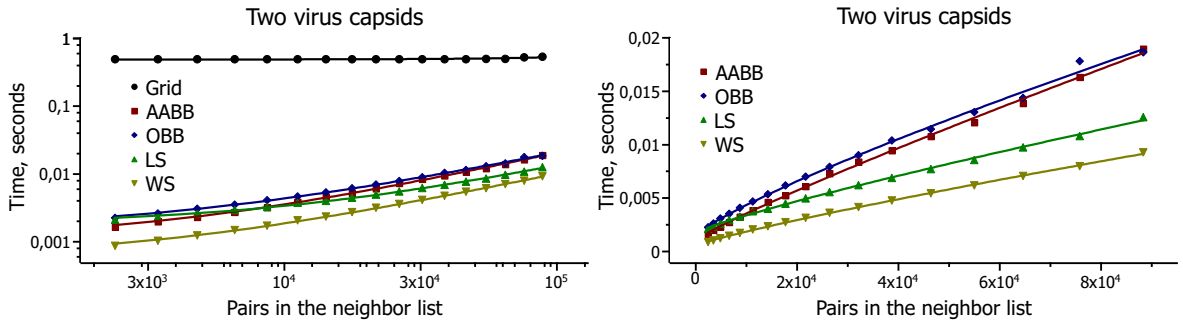


Figure 2.3.7: Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact for two bluetongue virus capsids in logarithmic scale (left), and of only hierarchy-based algorithms in linear scale (right). Solid lines represent fitting curves.

wide range of neighbor list sizes as cutoff varies. The resulting timings for $m = 8$ (AABB, OBB) and $m = 4$ (LS, WS), and cutoff varying from 4 to 13.5 Å with a step of 0.5 Å, are shown in Fig. 2.3.8 (left). All hierarchy-based algorithms depend on the number of pairs in the neighbor list in the same way as before. The grid method is outperformed by algorithms based on hierarchies of spheres, although all timings are very similar.

We also modified the last test: for any given cutoff, we applied random rotations to the second molecule. Running times of the algorithms are plotted as functions of the number of contacting pairs in Fig. 2.3.8 (right). Here, timings for the AABB-based algorithm are very dispersed (to show this, we plot average timings and indicate minimal and maximal values with error bars). This happens because after an AABB is rotated, it must be made axis-aligned again as described in Ref. [76], and sometimes the rotated AABB is being unnecessarily enlarged. Moreover, all boxes were enlarged by the half of the cutoff distance during the hierarchy construction, as described in Section 2.2.2. Therefore, a lot of unnecessary overlaps between boxes occur, which results in larger timings.

2.3.4 Distance test

In the second type of tests (distance test) we place two molecules of interest at some distance, and then move one of them towards the other in discrete steps. More precisely, we shift the second molecule of the complex (Table 2.1) 20 times by the same value along the x-axis. The cutoff is fixed for this type of tests.

To describe the benchmarks we define a *separation distance* between molecules as a minimal distance between all pairs of atoms such that one atom in this pair belongs to the first molecule of the complex and the other atom belongs to the second molecule.

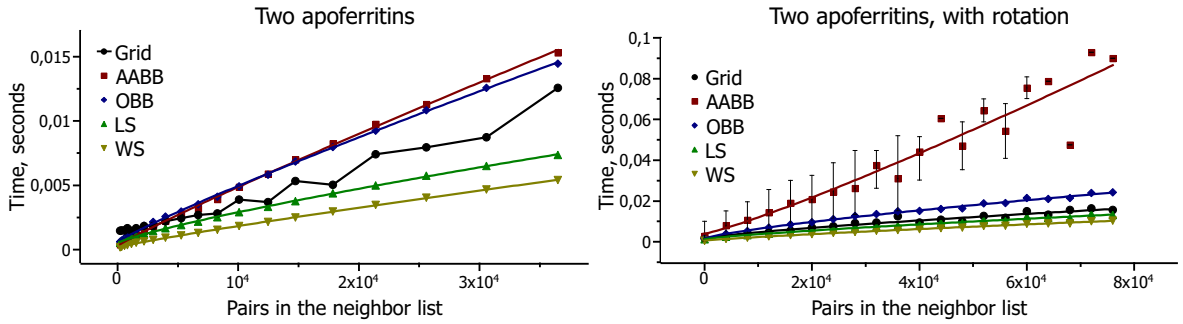


Figure 2.3.8: Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact without rotations (left) and with rotations (right) of one of the two apoferritin molecules. Solid lines represent fitting curves, except for the one corresponding to the grid algorithm timings on the left plot. Error bars on the right plot represent maximal and minimal values.

2.3.4.1 Biological complexes

First, we performed the distance test as described above for the Sa-barstar complex and the 70s ribosome, so that the separation distance between molecules was changed from 6.88 Å to 1.36 Å, and from 9.56 Å to 0.69 Å, respectively ($m = 8$ for AABB and OBB, $m = 4$ for spheres, cutoff distance 12 Å). The results are demonstrated in Fig. 2.3.9 on the left and right plots, respectively. As expected, timings for all algorithms based on the hierarchy of bounding volumes depend on the number of pairs in the neighbor list N as $c_i N^{a_i} + b_i$ ($a_i < 1,\ i = 1..4$). The time for the grid method depends on the number of ligand atoms that are mapped to non-empty grid-cells, on the number of atoms in each grid-cell to perform the comparison, etc. In other words, it depends on the shape of the ligand and receptor molecules. This time increases with the growing number of pairs in contact, because if the two molecules are closer in space, then more atoms of the ligand are mapped to grid cells, and more queries for interactions in neighbor cells and the current cell should be performed. It is clear from the plots that the grid method is outperformed by the sphere-based algorithms for a small complex, and by all hierarchy-based algorithms for a relatively large ribosome. The algorithm based on wrapped spheres, again, shows the best timings among all hierarchy-based algorithms.

2.3.4.2 Two bluetongue virus capsids

Then, we performed the distance test for two bluetongue virus capsids (cutoff is 12 Å, $m = 8$ for AABB's and OBB's, $m = 4$ for spheres). The separation distance between two replicas of the molecule was changed from 3.93 Å to 0.36 Å. The resulting timings as functions of the

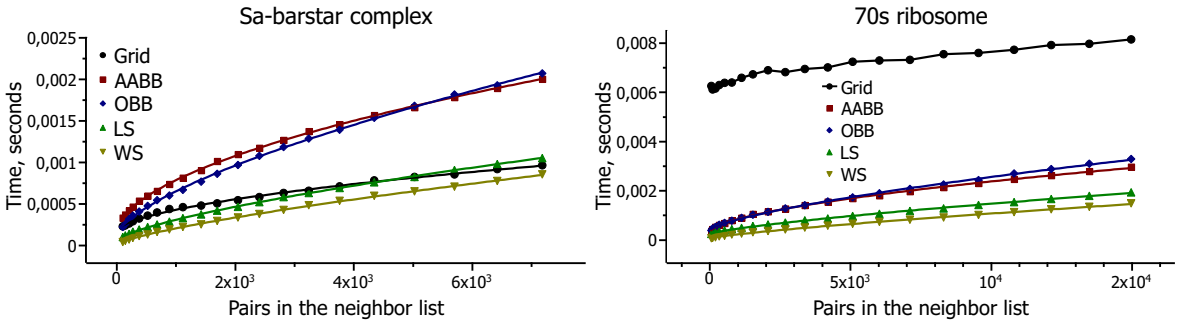


Figure 2.3.9: Distance test. Computational performances of all algorithms as functions of the number of pairs in contact within the fixed cutoff for Sa-barstar complex (left) and 70s ribosome (right). Solid lines represent fitting curves, except for the one corresponding to the grid algorithm timings on the right plot.

number of pairs in contact are shown in Fig. 2.3.10: in the logarithmic scale, for all algorithms on the left, and in the linear scale for only hierarchy-based algorithms on the right. All solid lines on the plots represent fitting curves of the type $c_i N^{a_i} + b_i$ ($a_i < 1,\ i = 1..5$). Again, for a system of two large rigid molecules, hierarchy-based algorithms clearly outperform the grid method. However, on the logarithmic scale we can see that at some point the lines have to cross. To obtain such a high number of pairs in contact, though, the two molecules have to interpenetrate significantly (which is a non-realistic molecular configuration) or the cutoff has to be large (which is not reasonable: the choice of the cutoff was discussed above).

2.3.4.3 Two apoferritin molecules

Finally, as in the previous section, we demonstrated the test results obtained for an apoferritin (Fig. 2.3.11, left), where we used two replicas of the same molecule. Solid lines on the plot represent fitting curves. Here cutoff was set to 12 Å, m for AABB- and OBB-based algorithms equal to 8, for sphere-based algorithms to 4. The separation distance between two replicas of the molecule was changed from 14.43 Å to 0.30 Å. The grid method is outperformed by hierarchy-based algorithms if the number of interacting pairs for two molecules is small. The fastest algorithm between hierarchical algorithms is the one based on the wrapped sphere hierarchy.

We can again slightly modify this test as follows. At each fixed distance (the cutoff is still constant) we perform several random rotations of one of the molecule replicas. All the algorithms are launched for each translation and rotation (described by a transform) and the time is measured (Fig. 2.3.11, right). As for the cutoff test, for the same reason as explained

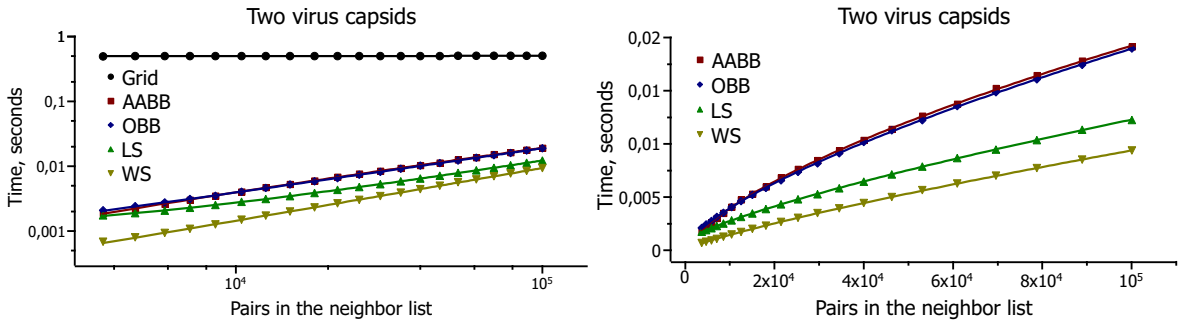


Figure 2.3.10: Distance test. Computational performances of all algorithms as functions of the number of pairs in contact for two bluetongue virus capsids in logarithmic scale (left), and for only hierarchy-based algorithms in linear scale (right). Solid lines represent fitting curves.

above, timings for AABB-based algorithm are dispersed. We indicate minimal and maximal values for these timings with error bars on the plot.

2.3.5 Remarks

It is important to mention that the exact timings will depend on the particular implementation of the algorithms *and* on the computer used to run the tests. This happens due to different strategies of memory allocations for different architectures, the cost of basic operations on the computer of use, cache size, cache speed, etc.

For example, in Fig. 2.3.12 we display the results of the cutoff test for the system of two apoferritin molecules without rotations with $m = 1$ for sphere-based algorithms, $m = 8$ for AABB- and OBB-based algorithms performed on Computer 1 (left plot, computer characteristics already described) and Computer 2 (right plot, Intel 2.40 GHz processor with 4GB of RAM, Windows Vista 32-bit operating system).

The general tendency for the algorithms complexities, though, remains the same: the complexity of hierarchy-based algorithms remains sublinear, the WS algorithm shows the best timings among all hierarchy-based algorithms and often outperforms the grid method.

Cache-efficient algorithms [184] (cache-oblivious [185, 186] or cache-aware) give an important improvement in the performance of the hierarchy-based algorithms. For example, some algorithms have been designed to compute cache-efficient layouts of bounding-volume hierarchies [187] or store these hierarchies in a smart way [188, 189].

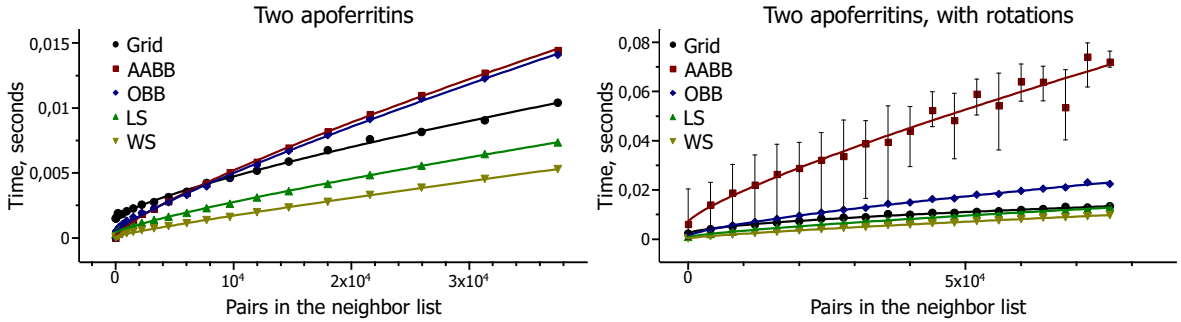


Figure 2.3.11: Distance test. Computational performances of all algorithms as functions of the number of pairs in contact without rotations of one of two apoferritin molecules (left) and with rotations (right). Solid lines represent fitting curves. Error bars on the right plot represent maximal and minimal values.

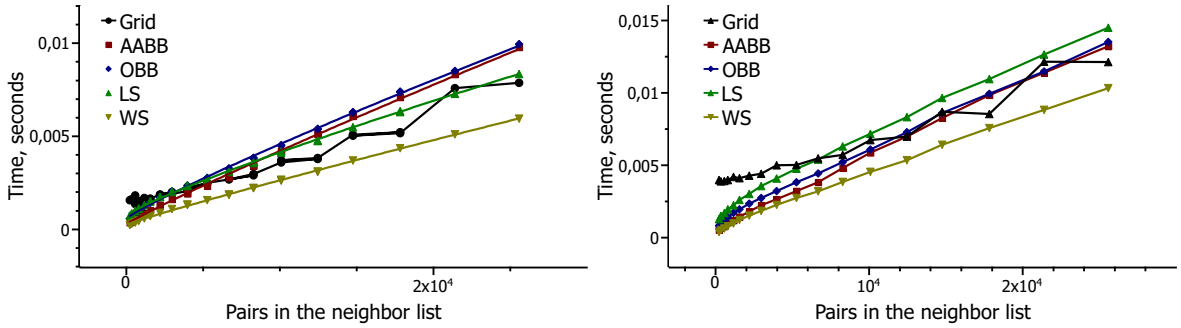


Figure 2.3.12: Cutoff test. Computational performances of all algorithms as functions of the number of pairs in contact for the system of two apoferritin molecules. Results obtained on Computer 1 (left) and on Computer 2 (right). Solid lines represent fitting curves, except for those for the grid algorithm.

2.4 Conclusion

In this Chapter, we have investigated the use of several algorithms to efficiently determine pairs of neighboring atoms within a certain cutoff distance when the molecules are rigid. More precisely, we have compared the grid data structure to four different types of bounding-volume hierarchies: the axis-aligned bounding box hierarchy, the oriented bounding box hierarchy, the layered sphere hierarchy and the wrapped sphere hierarchy. We have analyzed the performance of all these data structures based on several parameters: the size of the molecules, the average distance between them, the cutoff distance, as well as the type of bounding volume used in the culling hierarchy. We have demonstrated that, although slower to initialize, hierarchy-based neighbor search algorithms may in several cases perform more efficiently than grid-based algorithms for large rigid molecules. This happens, for example, if the number of pairs in contact for these molecules is relatively small compared to the size of the molecules, or when a small cutoff distance is considered.

In the future, we would like to study the possibility of designing a hybrid algorithm that would combine grids and hierarchies of bounding volumes. For example, a hybrid algorithm could first rapidly eliminate large irrelevant groups of atoms using hierarchies of bounding volumes, and would then locate neighboring pairs in the refined search region using the grid method.

Finally, we note that neighbor search for large rigid molecules may also be useful for Monte Carlo simulations of molecular self-assembly [190] and rigid body molecular dynamics simulations [191]. These applications could as well be investigated.

Appendix

Here we provide an analysis on the choice of the optimal leaf size parameter m for neighbor search algorithms based on hierarchies of bounding volumes. We present the results for a simple case: a system with a constant density, where both molecules in the complex contain N atoms and are in contact through their surfaces only.

Let us assume that the system configuration (*i.e.* molecular positions and orientations) is fixed and the cutoff distance for neighbor search is chosen. The goal is to find a value of parameter m that minimizes the time t of algorithms execution. This time contains two contributions. The first contribution is the time to descend to a leaf level $l = \log_2 \frac{N}{m}$ of the hierarchy, *i.e.* to perform all necessary algorithm operations down to the level l. This time increases with growing l as we perform more operations. The second contribution is a time to compute distances for all pairs of atoms inside leaf bounding volumes that are closer than the cutoff distance (BV's in contact). This time is proportional to m^2 and decreases with increasing l (since the leaf bounding volumes fit more and more closely the molecules, the number of pairs of atoms unnecessarily tested for proximity decreases). Therefore there may be one or more values of the level l for which t is a local minimum.

Thus, to determine a good value of l, we express the derivative of $l \mapsto t(l)$ as a finite difference $t(l+1) - t(l)$, and find its roots. In other words we set equal the two complexities $t(l)$ and $t(l+1)$. These are the algorithm's time complexities for two pairs of hierarchies constructed for the same system: in the first pair there are m atoms in each leaf node (hence, the trees heights are equal to $l = \log_2 \frac{N}{m}$), and in the second pair there are $m/2$ atoms in each leaf node (and the trees heights are equal to $l+1$).

The algorithm traverses each pair of hierarchies in a top-down manner, and its workflow is the same on both pairs until it reaches level l of the tree. Therefore, we will only compare the algorithm's costs after reaching this level. For the pair of trees with height l, this cost is the one of computing distances for all pairs of atoms inside leaf bounding volumes that are in contact (*i.e.* closer than the cutoff distance). For the pair of trees with height $l+1$, this cost is the sum of the cost of descending one more hierarchy level using the algorithm, plus the cost of treating the leaf bounding volumes.

Let us denote by c_a (resp. c_{bv}) the cost of determining whether two atoms (resp. two bounding volumes) are closer than the cutoff distance, and let $P(l)$ (resp. $P(l+1)$) be the number of pairs of bounding volumes in contact at level l (resp. $l+1$). Then, for the first pair of hierarchies, the cost of determining pairs of neighboring atoms after reaching level l is the following:

$$P(l)m^2c_a.$$

For the second pair of hierarchies, however, proceeding from level l to level $l+1$ costs $4P\left(l\right)c_{bv}$, since each bounding volume in contact at level l is divided into two bounding volumes, while treating the leaf bounding volumes at level $l+1$ costs $P(l+1)\left(m^2/4\right)c_a$. The whole complexity in this case is:

cutoff, Å	AABB, average r	value of m	WS, average r	value of m
4	1.66	6.96	1.74	3.23
6	1.81	7.19	1.82	3.28
8	1.93	7.4	1.91	3.35
10	2.03	7.58	1.99	3.42
12	2.13	7.78	2.07	3.49

Table 2.3: Experimentally obtained values for the average ratio r for different values of the cutoff and two types of bounding volumes. Corresponding values of parameter m.

$$4P\left(l\right)c_{bv}+P\left(l+1\right)\frac{m^2}{4}c_a.$$

Thus, the equation $t(l)=t(l+1)$ can be written as follows:

$$P(l)m^2c_a=4P\left(l\right)c_{bv}+P\left(l+1\right)\frac{m^2}{4}c_a. \tag{2.4.1}$$

Let us now estimate the ratio $r=P\left(l+1\right)/P(l)$. Only some of the four possible pairs of child bounding volumes for each pair in contact may remain closer than the cutoff. Therefore, r is between zero and four.

In practice, this ratio depends on a large number of parameters (system's characteristics, tree properties, etc.), and it is difficult to describe it analytically. Thus, we experimentally determined some average values for r. To do this, we constructed hierarchies of AABB's and WS's with only one atom in each leaf node (which resulted in a maximal tree height) for the 70s ribosome (Fig. 2.3.1). Then, for both types of bounding volumes, and for cutoff values varying from 4 to 12 Å with a step of 2 Å, we measured the ratios $r(l)=P(l+1)/P(l)$ and averaged them over all hierarchy levels. The results are summarized in Table 2.3.

The ratio c_{bv}/c_a for AABB's and WS's (7.08 and 1.47, respectively) was computed using the information from Section 2.2.2.

Substituting numerical values for these two ratios in equation (2.4.1), we find corresponding values of m (also shown in Table 2.3). These numbers are in good agreement with the numerical results obtained above: 8 and 4 for AABB's and WS's, respectively.

Chapter 3

Fast construction of hierarchical representations for molecular graphs

Comme nous l'avons démontré dans le Chapitre précédent, les algorithmes de recherche des voisins qui sont fondés sur des représentations hiérarchiques peuvent être bénéfiques pour certains systèmes moléculaires. Ces représentations sont également utilisées par d'autres algorithmes de modélisation et simulation. Comme les topologies des systèmes moléculaires peuvent être complexes, la construction automatique de ces décompositions hiérarchiques est parfois difficile. Jusqu'à présent, aucune stratégie générale de construction n'a été proposée.

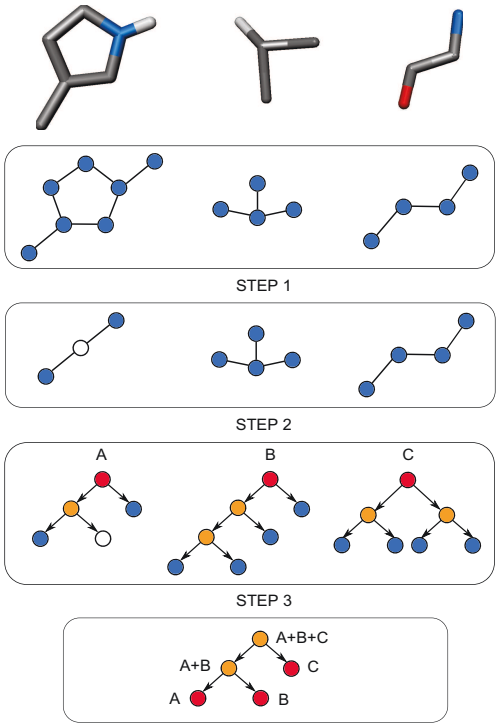
Dans ce Chapitre, nous présentons un algorithme rapide et général pour la construction complète d'une représentation hiérarchique d'un système moléculaire. Cet algorithme en trois étapes traite le système moléculaire d'entrée comme un graphe dans lequel les sommets représentent des atomes ou des pseudo-atomes, et les arêtes des liaisons covalentes. La première étape fusionne tous les cycles dans le graphe d'entrée. La seconde étape construit un arbre à partir du graphe réduit. La troisième étape construit un arbre final à partir des arbres représentant les composantes connexes du graphe.

Nous analysons la complexité de cet algorithme et démontrons ses performances sur un ensemble de cas tests difficiles, ainsi que sur un grand sous-ensemble de graphes moléculaires extraits de la Protein Data Bank. En particulier, nous démontrons expérimentalement que les deux premières étapes sont linéaires en fonction du nombre d'arêtes dans le graphe d'entrée (le facteur de branchement est fixé pour la deuxième étape). Nous démontrons aussi que pour les systèmes répartis de manière homogène, la troisième étape est capable de construire un arbre en temps linéaire en fonction du nombre de points d'entrée. Finalement, nous démontrons une application de notre algorithme pour la mécanique moléculaire adaptative en angles de torsion.

As has been shown in the previous Chapter, neighbor search algorithms relying on hierarchical representations may be beneficial for some molecular systems. These representations are also being used by other modeling and simulation algorithms. Given the potentially complex topologies of a molecular system, though, automatically generating such hierarchical decomposition may be difficult. Up to now, no general strategy of building a good tree was proposed.

In this Chapter, we present a fast general algorithm for the complete construction of a hierarchical representation of a molecular system. This three-step algorithm treats the input molecular system as a graph in which vertices represent atoms or pseudo-atoms, and edges represent covalent bonds. The first step contracts all cycles in the input graph. The second step builds an assembly tree from the reduced graph. The third step builds a final tree from the trees representing connected components of the graph.

We analyze the complexity of this algorithm and demonstrate its performance on a set of specifically tailored difficult cases, as well as on a large subset of molecular graphs extracted from the Protein Data Bank. In particular, we experimentally show that the first and the second steps behave linearly in the number of edges in the input graph (the branching factor is fixed for the second step). We also experimentally show that for homogeneously distributed systems, the third step is able to build a tree in linear time in the number of input points. Finally, we demonstrate an application of our hierarchy construction algorithm to adaptive torsion-angle molecular mechanics.



3.1 Introduction

In the previous Chapter we have compared several neighbor search algorithms, and we have demonstrated that for the simulations, where large rigid blocks of atoms (molecules) are considered, the hierarchically-based approach is beneficial.

Although hierarchical representations are widely used by many other modeling and simulation algorithms as, for example, simulations in internal coordinates [40, 21, 22, 41, 42], no efficient way to construct such a representation from an arbitrary molecular graph in the general case has been proposed.

In this Chapter, we introduce a fast algorithm for building a hierarchical representation (assembly tree) from a *molecular graph*. In this graph, vertices may be atoms, pseudo-atoms or higher-level descriptions of molecular entities; graph edges, if present, may respectively represent covalent bonds, pseudo-bonds or generic connections between graph vertices; and graph connected components may be associated with distinct molecules. An example of a molecular graph corresponding to a molecular system is shown in Fig. 3.1.1.

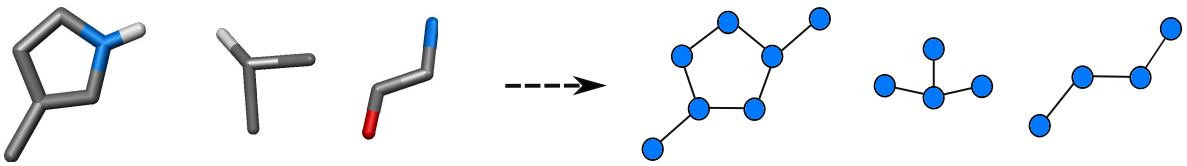


Figure 3.1.1: Example of a molecular system (left) and a corresponding molecular graph (right). Graph vertices are atoms, graph edges are covalent bonds, graph connected components are molecules.

By an *assembly tree corresponding to a graph* we denote a binary tree in which each leaf node refers to an individual graph vertex, and each internal node represents a sub-graph that is obtained by connecting together the sub-graphs referred to by its children. For example, in Fig. 3.1.2 we display a simple graph and a tree corresponding to this graph.

For our purposes, we want the constructed assembly tree to possess two main properties: (a) topologically-connected parts of the graph should be close in the tree; (b) the resulting tree should be as balanced as possible. Both properties are crucial for *e.g.* neighbor-list construction algorithms (see [76] and references therein) and balance is helpful when the algorithm is implemented on a parallel architecture. We also obviously want the tree construction algorithm to be fast and efficient.

The algorithm that we propose for constructing an assembly tree for a molecular graph consists of three steps:

1. **Cycle contraction**: in the general case, the graphs we are dealing with may contain cycles (*e.g.* aromatic rings), including cycles-within-cycles. To build a binary assembly tree, the underlying graph needs to be acyclic: the cycles should be broken or contracted. For simulations in internal coordinates and their applications, cycle contraction is preferable. Therefore, the first step of the algorithm traverses the original molecular graph

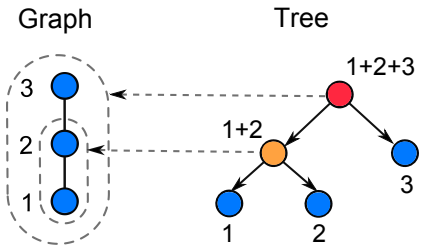


Figure 3.1.2: Assembly tree corresponding to a graph. Blue nodes represent vertices of the input graph and the tree nodes corresponding to them (they have indices from 1 to 3). Yellow nodes describe internal nodes of the tree. The red node is the root node of the tree. Internal nodes and the root node are marked according to the content of the corresponding sub-graph: for example, the node marked "1+2" corresponds to a sub-graph containing vertices 1 and 2. Dashed grey arrows indicate sub-graphs corresponding to the internal node of the tree and the root node.

and contracts all the cycles it contains, which produces a *contracted graph*. We let the user decide to break some cycles before executing the algorithm if necessary, depending on the application[1].

2. **Assembly tree construction for a connected component**: the second step builds an assembly tree for each contracted connected component of the graph.

3. **Assembly tree construction for the whole molecular system**: the third step gathers all trees representing connected components to form a final tree.

The rest of the Chapter is organized as follows. First, we introduce some basic definitions used throughout the Chapter and formalize the problem. Then we present our algorithm in detail and analyze its properties. Finally, we provide experimental results, and show possible applications of the algorithm.

3.2 Basic definitions and problem statement

Let us introduce some notations.

An *undirected graph* is an ordered pair $G = (V, E)$, where V is a finite, non-empty set of *vertices,* and E is a finite set of *edges*: pairs of vertices (edge *ends*). An edge and its ends are *incident* to each other. Two ends of an edge are *adjacent*. The *degree* of a vertex is the number of edges incident with it. Vertex of degree one is a *pendant* vertex, and its incident

[1]For example, the user will probably choose to model disulphide bonds by supplementary terms in the potential energy function, and not with kinematic constraints (hard constraints on the distance), so that long cycles that would result from disulphide bonds would not be contracted. Further discussion on how and when to break cycles in the general case can be found in Section 3.5.

edge is a *pendant* edge. The *branching factor* of the graph is the biggest vertex degree among all vertices.

A *path* is a sequence $(v_0; e_0; \ldots; e_{n-1}; v_n)$, where $e_i \in E$ for all i has ends v_i and v_{i+1}, and all vertices (except perhaps the first and the last) are distinct. A *cycle* is a path of positive length whose first and last vertices are equal. A *loop* is an edge whose two ends are equal. A graph has *multiple edges* if it has two different edges with the same ends. A graph is *connected* if there is a path from any one vertex to any other. An edge is called *cycle's external edge* if it does not belong to the cycle but is incident to some vertex from this cycle.

A *connected component* of a graph is a connected sub-graph of a graph.

Graph contraction is a procedure of substituting every cycle in the graph by a single vertex (*group vertex*), so that graph connectivity is preserved: group vertex is incident with all cycle's external edges. After contraction, the graph becomes *contracted*.

Binary tree is a tree data structure in which each *node* (*parent*) has two *children* (*left* and *right*) or no children at all. The *root node* of a tree has no parents. A *leaf node* has no children. Nodes that have a parent and two children are called *internal nodes*. The *depth* of a node is the length of the path from the root to the node through internal nodes. The set of all nodes at a given depth is a *level* of the tree. The *height* of a tree is the length of the path from the root to its deepest node.

We can now precisely formulate the **Problem**. Let G be a graph which may contain cycles and multiple connected components, but neither loops nor multiple edges. The problem is to build an assembly tree corresponding to the contracted input graph.

3.3 Algorithm description

The algorithm that we propose consists of three steps. The input and the output for the first and the second steps of the algorithms are shown in Fig. 3.3.1: algorithm's input is a graph with cycles, the output of the first step is a contracted graph, and the output of the second step is an assembly tree for the contracted graph. The structure of the complete three-step algorithm is shown in Fig. 3.3.2. The output of the third step of the algorithm is a binary tree, corresponding to the whole molecular graph comprising several connected components.

During the first and the second step of the algorithm, we might make several *passes* through the graph, *i.e.* traverse the whole graph or its sub-graphs. Therefore, the graph traversal should be efficient. To achieve this, the depth-first search algorithm is used [192]: we only move forward along the edges that have a non-visited vertex on the other end. If there are no such edges, we go backward, until we find a vertex with non-traversed edges. If there are no such vertices, the algorithm terminates. Traversing a graph this way, each edge is visited *at most two times.*

3.3.1 Contracting cycles

We start the algorithm description from its first step: detecting and contracting cycles in the input graph.

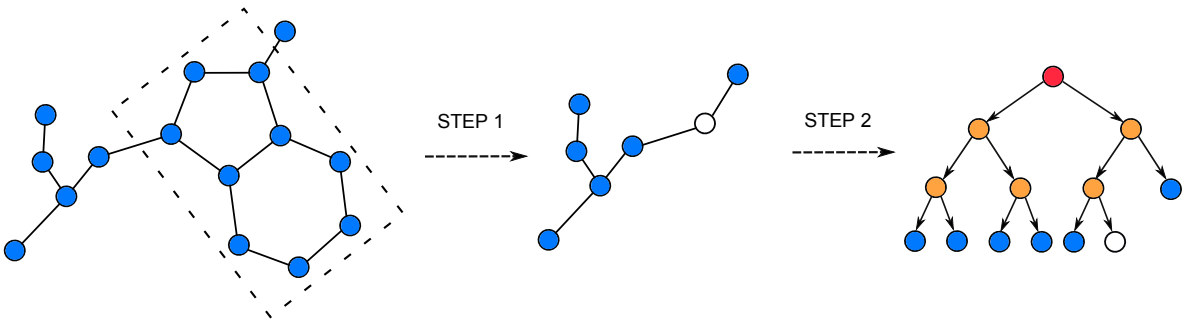


Figure 3.3.1: Workflow of the first and the second steps of the tree construction algorithm, overview. Blue nodes represent vertices of the input graph (input for steps 1 and 2), and tree nodes, corresponding to them (output of step 2); white nodes stand for group vertices (output of step 1) and tree nodes, corresponding to them (output of step 2); yellow nodes denote internal nodes of the final tree, the red node corresponds to the root node of the tree. Dashed lines indicate the cycles to be contracted. Relationship between parent and child nodes in a tree is indicated by arrows.

Several authors have addressed related problems. For example, Mateti et al. [193] give an overview of the algorithms to enumerate all cycles in a graph. Of all the algorithms analyzed, the asymptotically fastest algorithm has an upper bound of $O((V+E)C)$ on the number of operations (C is a number of so-called elementary circuits, V and E are numbers of vertices and edges). For detecting all cycles in the graph parallel computing may be used [194][2].

The algorithm proposed here consists of two passes through the graph. First, we detect and mark every edge that belongs to some cycle. We do it using the linear-time algorithm by Tarjan [192][3]. Then, we contract all *marked connected components* (*i.e.* sets of vertices connected by marked edges only). We replace each marked connected component by a group vertex. This group vertex will be incident with zero or more non-marked edges, having as their ends the vertices from the marked connected component. We traverse all marked connected components of the graph and make the contraction on the fly. Before starting the traversal of the marked connected component, we create a new group vertex, which will represent this component. Then, we begin a depth-first search for the graph from a randomly picked vertex v^* of this component. If an edge we picked next is marked, we move through it. If the next edge is not marked, we connect it to the new group vertex: we assign the new group vertex

[2]Some authors have addressed the related contractability problem: a graph G is H-contractible if H can be obtained from G by a sequence of edge contractions [195], [196]. This is not the problem we are studying, although the input graph of this step is contractible to its output graph.

[3]Finding the set of edges that belong to at least one cycle is equivalent to finding the set of *bridges* (an edge is called a *bridge,* if by removing it from the graph, we increase the number of connected components). To do it we use the same data structures as described by Tarjan for finding the set of articulation points. An edge traversed by depth-first search from vertex v_1 to a vertex v_2 is a bridge if and only if nums[v_2]=lows[v_2] and lows[v_1] < lows[v_2].

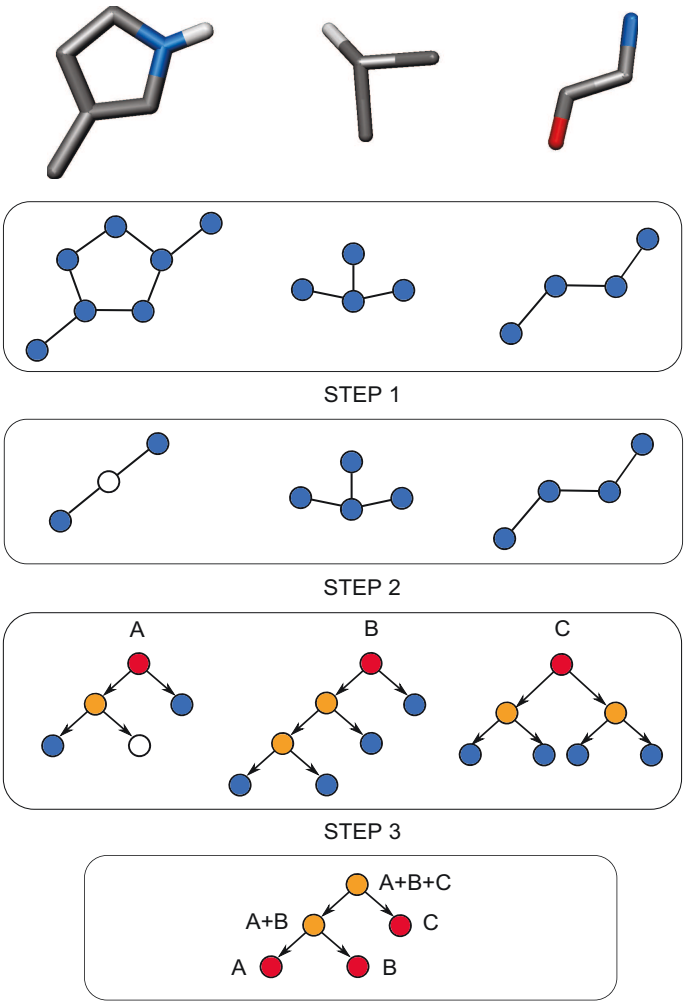


Figure 3.3.2: The workflow of the complete tree construction algorithm for the general molecular graph containing several connected components. Blue nodes represent vertices of the input graph (input for steps 1 and 2), and tree nodes, corresponding to them (output of step 2); white nodes stand for group vertices (output of step 1) and tree nodes, corresponding to them (output of step 2); yellow nodes denote internal nodes of the final tree; red nodes represent root nodes of the trees corresponding to connected components of the graph (output of step 2 and input for step 3). Relationship between parent and child nodes in a tree is indicated by arrows.

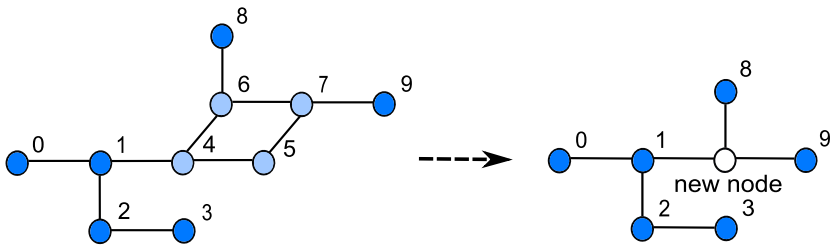


Figure 3.3.3: Workflow of the first step of the tree construction algorithm, example of a cycle contraction.

as its appropriate end, and then we work with the next non-visited edge of the same vertex v^* (we do not traverse the non-marked edge).

This procedure produces a completely acyclic graph, where every marked connected component is replaced by a single vertex. Description of this group vertex can contain a list of vertices that the group vertex is representing. Contracted cycles will be represented in the assembly tree by corresponding group vertices. Figure 3.3.3 gives an example of this algorithm for a simple case. Please note that after the first step of the algorithm all connected components of the graph remain connected.

3.3.2 Building assembly trees for connected components

We can now describe the second step of the algorithm: building an assembly tree for an acyclic connected graph. Precisely, nodes in this tree represent parts of the connected component: leaf nodes correspond to vertices, and internal nodes correspond to unions of two sub-graphs connected by an edge. In this step group vertices will be treated as all other non-group graph vertices.

We construct the assembly tree based on the graph connectivity. The most obvious procedure would be the recursive splitting of the graph in halves. But to do that we have to traverse the graph (or its sub-graphs) each time before splitting. That is why the whole algorithm would have $O(N \log N)$ complexity, though the obtained tree would be well-balanced.

The basic idea behind our algorithm is very simple, and consists in repeating the following two passes. First, we mark as many edges as we can during a graph traversal, such that no marked edge shares a vertex with another marked edge. Second, we contract the ends of each marked edge in parallel with gathering into a new tree node the tree nodes corresponding to the edge ends. The iterations stop when only one vertex is left in the graph. This final vertex corresponds to the root node of the assembly tree of the connected component.

The problem is that, in this case, we do not guarantee avoiding the situation[4] like the one shown in Fig. 3.3.4. Here, if at each iteration we traverse a graph in a specific order (from vertex 1 to 5), one pendant edge is not contracted until the very end, which results in a

[4]That would be, for example, the case for every chain of length $2^n + 1$.

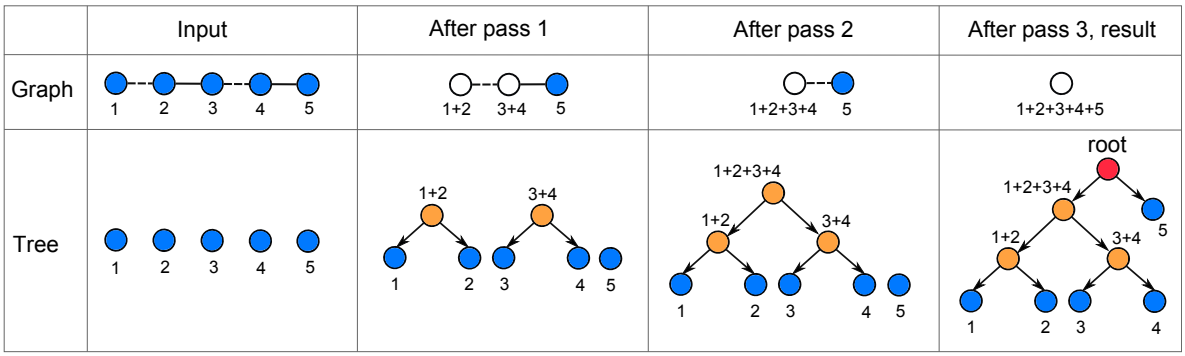


Figure 3.3.4: The second step of the tree construction algorithm, construction of an unnecessarily unbalanced tree using non-modified algorithm. Blue nodes represent vertices of the input graph and tree nodes, corresponding to them; white nodes stand for vertices representing contracted sub-graphs and correspond to the internal nodes of the tree; yellow nodes denote internal nodes of the final tree; the red node represents the root node of the tree. Dashed edges are the edges to be contracted during the next pass.

tree that could be better balanced[5]. That is why, to build a better balanced tree, we should modify the algorithm so that pendant edges are contracted as soon as possible. Of course, it is not possible to create a balanced tree for any input graph. For example, for a star-like graph, the tree will always be unbalanced (see Fig. 3.3.5), but can be improved using a *link splitting* technique described in Ref. [22].

Let us describe an iteration of the modified algorithm in more detail.

In the modified algorithm, at each iteration, we still perform two passes through the graph. During the first one we mark not only all the edges that are marked in the basic version, but also pendant edges. As a result, we have several marked connected components in a graph. We will traverse and contract them during the second pass as we did in 3.3.1.

While traversing a marked connected component, we construct a corresponding sub-tree as follows. From two nodes, corresponding to the first two visited vertices, a tree node is created, called *current node*. Then, anytime we traverse a new vertex v, we gather the current node with the node representing this vertex v into a new node. This new node becomes the current node. Please note that, anytime we gather two nodes, the corresponding vertices have a connecting edge. This guarantees that the sub-tree is valid. We show the workflow of the modified algorithm in Fig. 3.3.6.

3.3.3 Building the assembly tree of the complete graph

The third step of the algorithm consists in building the final assembly tree from the output of the second step: assembly trees associated with the connected components of the graph.

[5] Even if we start each graph traversal from a random vertex we cannot guarantee avoiding this case.

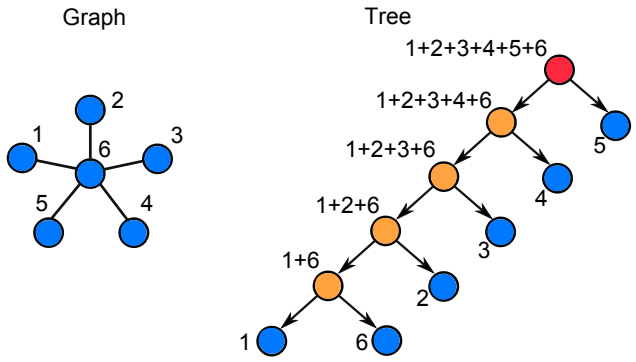


Figure 3.3.5: The second step of the tree construction algorithm, a big branching factor example: the tree is always unbalanced. Blue nodes represent vertices of the input graph, and tree nodes, corresponding to them; yellow nodes describe internal nodes of the final tree; the red node represents the root node of the tree.

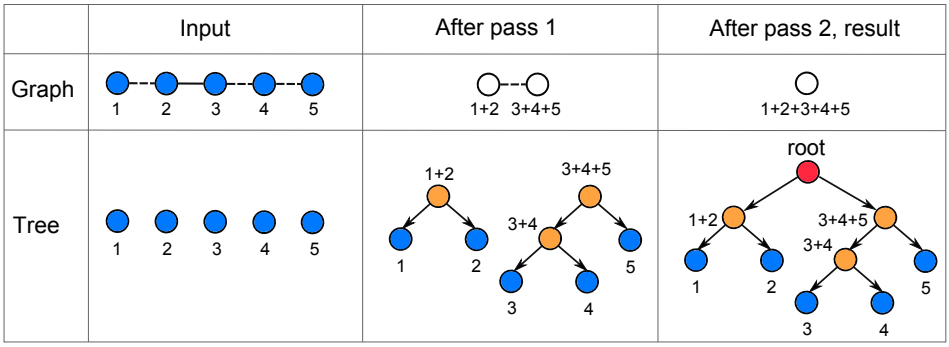


Figure 3.3.6: The second step of the tree construction algorithm, construction of an assembly tree using modified algorithm. Blue nodes represent vertices of the input graph and tree nodes, corresponding to them; white nodes stand for vertices representing contracted subgraphs and correspond to the internal nodes of the tree; yellow nodes denote internal nodes of the final tree; the red node represents the root node of the tree. Dashed edges are the edges to be contracted during the next pass.

From these trees, we only retain their root nodes, and characterize each root node by a point in k-dimensional ($k = 2, 3$) space: a natural choice may be the geometric center (if all vertices have identical weight) or some weighted average of the positions of vertices in the initial graph (*e.g.* gravity center).[6]. The problem of tree construction for a geometrical system is very important in N-body simulations including long-range interactions in molecular simulations and celestial mechanics. It has already been studied by a number of authors in different domains, for example, in Ref. [176, 197, 198], but the algorithms proposed there have a $O(N \log N)$ best-case complexity in the number of bodies in the system. We propose a novel, faster bottom-up algorithm that relies on space-filling curves.

First, we determine the smallest axis-aligned parallelepiped that bounds all nodes, and associate a hierarchy of cells to it as follows. Along each dimension this bounding parallelepiped is divided into p parts, so that we obtain p^k parallelepipedic cells of equal dimensions. If 2^h is the closest superior to p power-of-two integer, then the height of the hierarchy is h. The parent for the current cell is a parallelepipedic cell of the upper layer, which has coordinates two times smaller along each axis and which is twice larger in all dimensions. The children of the cell are all the cells that consider it as a parent. For example, in Fig. 3.3.7 we show two levels of the hierarchy of cells in 2D: m and $m-1$. Coordinates of the orange cell on the level m are (1,1) (see the marks on the parallelepiped border, the smaller number is considered). Therefore, to obtain the coordinates of its parent cell we divide its coordinates by two: $1/2 = 0$ as we search for integer numbers. Thus, the parent cell for the orange cell is the one with coordinates (0,0) at the level $m-1$ and it is colored in orange too. For the green cell at the level m (coordinates (2,2)) the parent cell is the green one at the level $m-1$ (coordinates (1,1)).

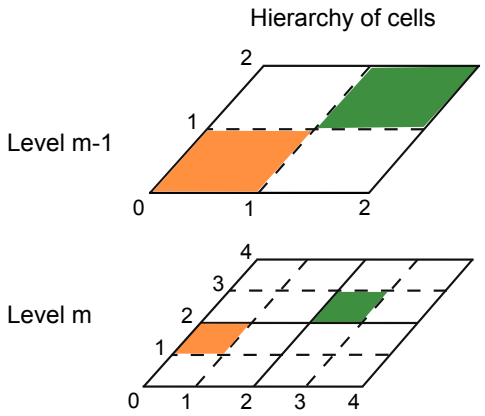


Figure 3.3.7: Hierarchy of cells in 2D. Parent-child relationship between cells. Detailed description in the text.

Then, this hierarchy of cells is used to build the assembly tree as described below.

During the initialization step, we distribute all input nodes to the cells of the deepest hierarchy layer (h-th layer) according to their position in k-space. These cells should contain a small number of nodes (one or two is better). For each non-empty cell, we assemble the nodes inside it into a *random* binary tree. For root node of this tree, we compute the *space-filling curve index* (SFC index) i of the current parallelepiped ($1 \leqslant i \leqslant 2^k$), obtained as

[6]The resulting trees may depend on this choice, and some may be more suitable than the others: for example, for neighbor search problems the geometric center is preferable. We leave this decision based on the particular application to the reader, as it is not a part of this study.

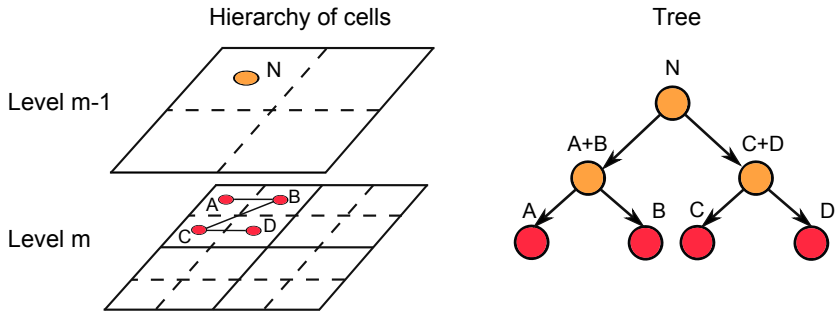


Figure 3.3.8: The third step of the tree construction algorithm, algorithm iteration for a non-empty cell in 2D. Red nodes represent root nodes of the sub-graphs corresponding to the connected components of the input graph; yellow nodes describe internal nodes of the final tree.

follows. If we take the $(h-1)$-layer parent cell of the current cell, and we traverse its j children ($1 \leqslant j \leqslant 2^k$) along a space-filling curve in the k-dimensional space (*e.g.* a Peano curve, [199, 200]), the SFC index of the current cell would be its position in this traversal sequence. Finally, the obtained root node is put to the parent cell so that its children are stored inside in the order of their SFC indices (see Fig. 3.3.8).

Then, the following iteration of the algorithm is repeated, until we have only one non-empty cell for the current layer. If we worked with the m-th hierarchy layer, we move to layer $m-1$. For each non-empty cell of this layer, we build a binary tree from the contained nodes, respecting their order[7]. As before, the SFC index is calculated for the resulting node, and this node is put to the parent cell. For the last cell we connect the nodes inside it according to their SFC indices. The complete assembly tree has then been built. You can see an example of an algorithm iteration on Fig. 3.3.8.

For convenience, we provide a summarized pseudo-code description of the complete algorithm (see Algorithm 3.1).

3.4 Algorithm analysis

In this section, we analyze the complexity of the proposed algorithm and discuss properties of the output of each step. We do not consider isolated graph vertices as input graphs: in this case a constant computational effort is required per each such vertex.

[7] If nodes are stored in a list inside the cell, we repeat the following procedure until there is only one node left in the current list. We assemble nodes in the current list pairwise (according to their order: the first node with the second one, the third node with the forth, etc.) and put newly obtained nodes to a new list. If the number of nodes in the current list is odd, the last node of this list is moved directly to the new list. Then the new list becomes the current.

Algorithm 3.1 Complete description of the algorithm for constructing an assembly tree from a molecular graph in pseudo-code.

STEP 1 (cycle contraction):
  a) pass 1, mark cycled edges
  b) pass 2, contract edges in marked connected components
STEP 2 (creating assembly trees for all connected components):
  current graph = contracted graph
  while number of vertices in the current graph >1 **do**:
    a) pass 1, mark every second edge and pendant edges
    b) pass 2, traverse marked components and contract them,
      make a subtree from nodes, corresponding to each marked component
    c) current graph = contracted graph
STEP 3 (building of the final hierarchical representation):
  a) find boundaries of the system, create hierarchy of cells
  b) put nodes representing components to the lowest layer of the hierarchy of cells
  c) **while** number of non-empty cells on current level > 1 **do**:
      for each non-empty cell of this level
        connect nodes inside cell, produce a subtree
        put an obtained node to the upper-level cell
      move to the upper level of the hierarchy of cells

3.4.1 Step 1: cycle contraction

The first step is straightforward. In this step, we perform two passes through the graph. The first pass (as a depth-first search [192]) is linear in $E + V$, where E is the number of edges and V is the number of vertices. The second pass is linear in $E_m + V_m$, where E_m and V_m are numbers of edges and vertices in marked connected components, because we traverse only them. Hence, the whole step has a linear complexity in the number of edges in the original input graph:

$$f_1(E) = O(E + V) + O(E_m + V_m) \leq 2O(E) = O(E). \tag{3.4.1}$$

3.4.2 Step 2: building an assembly tree for a connected component

In section 3.3.2, we saw that during the second step we perform several iterations on the graph, each one consisting of two passes through it.

At this step, we do not have cycles in the graph anymore. Therefore, a depth-first search at the first pass of each iteration is linear in the number of edges in the current graph. During the second pass, the contraction of marked connected components is linear in the number of edges they contain. This number is a sum of the numbers of directly marked edges and pendant edges. The former is inversely related to the branching factor B (for each vertex we mark only one incident edge). The latter is proportional to $-1/B$.

Supposing the input graph has E edges, the total complexity can be written as follows:

$$f_2(E) = O(E) + O(1/B). \tag{3.4.2}$$

For a fixed value of the branching factor, the second step of the algorithm is linear in the number of edges in the connected graph.

If we vary the branching factor of the system, the biggest running time corresponds to a simple chain. It happens because in this case we delete approximately half of the edges during each iteration, and strictly more than a half otherwise.

3.4.3 Step 3: building an assembly tree for the whole system

We now explore the third step of the algorithm.

Let us first analyze its time complexity and then its space complexity. Creating the lowest layer of the hierarchy of cells (finding maximal and minimal values for all coordinates of the nodes, dividing the total volume) and distributing nodes to these parallelepipeds is linear in the number of nodes. Then, on each iteration, for each non-empty cell, we assemble nodes inside the cell. Therefore, the overall complexity of the third step is dependent on the number of iterations necessary for the algorithm, *i.e.* on the spatial distribution of input nodes for this step. If this distribution is uniform or close to it, the number of iterations is $\log(N)$ and each iteration decreases the number of nodes by a factor of 2^k, so the whole algorithm is linear in the number of connected components N in the graph (see experimental results below):

$$f_3(N) = O(N). \tag{3.4.3}$$

When the geometrical distribution is not uniform, the number of iterations is larger than $\log(N)$ and the overall complexity may be as large as $O(hN)$, where h is the height of the hierarchy of cells.

In the worst case the amount of memory used by the algorithm is proportional to N^k, but it normally depends on the number of cells in the lowest layer of the hierarchy. This can be reduced if we do not allocate memory for all cells of each layer, but use a hash table and an appropriate hashing function. However, the complexity of the algorithm becomes dependent on the cost of inserting an element into the hash table.

Concerning the properties of the obtained tree, we do not guarantee the tree to be balanced, but this actually reflects the potentially non-uniform distribution of nodes in space[8]. Moreover, the nodes inside each cell are gathered according to a space-filling curve, so the overall overlap of left and right sub-trees (*i.e.* the overlap between their axis-aligned bounding boxes) is rather small. This is important for adaptive algorithms that rely (for efficiency) on having little interaction between subsystems [79].

[8] If a tree is well-balanced, then the system distribution is close to uniform. If there is a large difference between heights of left and right sub-trees, then corresponding cell volumes contain different number of connected components.

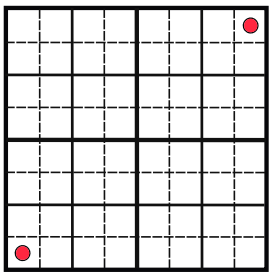


Figure 3.4.1: The third step of the tree construction algorithm, planar example: the chosen number of cells is too big. Red nodes represent root nodes of the sub-graphs corresponding to the connected components of the input graph. Different types of lines (bold, dashed) correspond to different levels of the hierarchy of cells.

Let us now discuss the choice of the number p of cells along each axis in the lowest layer of the hierarchy. This choice depends on the application domain. In molecular biology, for example, the fact that the length of the covalent bond cannot exceed 3.5 Angstroms may be used (see below). For other applications, dividing the longest axis by the number of connected components may be employed to obtain the required number of cells. That would be a good solution for a system that is uniformly distributed in space. If the incorrectly chosen number is too big, we can obtain a grid too fine as the one on the Fig. 3.4.1. In this 2D example, we perform several unnecessary iterations. If the chosen number is too small, we will have too many nodes randomly assembled into a tree during the initialization step. This problem has been studied by several authors. For example, in Ref. [201] the spheres are considered and they are inserted to the octree (top-down insertion) until there is the one sphere in the containing cell, and from the beginning, the lowest layer of the hierarchy is correctly chosen. In our applications, though, more complex structures are considered and bottom-up tree construction is used, so this approach is not completely applicable.

3.5 Implementation and results

The complete algorithm has been implemented in C++ into the GofAsTr (Generator of Assembly Trees) library, which is available through the NANO-D group website, and it tested (unless otherwise specified) on an Intel 2.40 GHz processor with 4GB of RAM, Windows Vista 32-bit operating system.

We used two categories of benchmarks to test our algorithm. The first category includes specifically tailored difficult cases for each step of the algorithm. The second one is a large subset of molecular graphs extracted from the Protein Data Bank [128].

3.5.1 Difficult cases

Let us first consider the benchmark comprising difficult cases for each step of the algorithm.

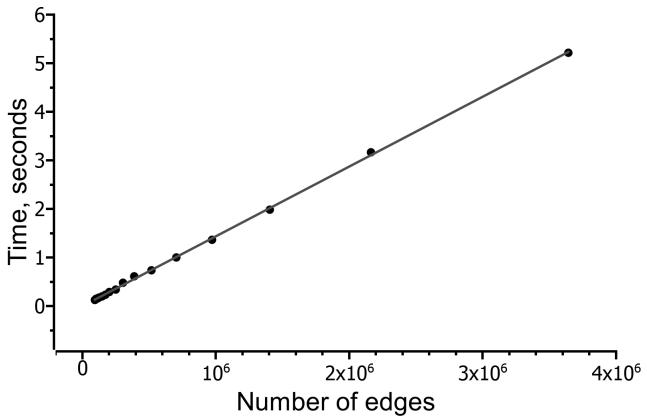


Figure 3.5.1: Computational performance of the algorithm's first step on finite cubical 3D lattices. Running time as a function of the number of edges in the input graph is plotted. The solid line represents a linear fit.

3.5.1.1 Step 1

To test the linearity of the first step of the algorithm, we designed the following benchmark: vertices are set to all node-points of a finite 3D cubical lattice, and every internal vertex of this lattice is connected to its eight closest neighbors. We varied the size of the lattice, which changed the number of edges in the graph, and measured the running time. As all edges here belong to cycles, both passes are linear in the total number of edges, and so is the whole algorithm. Fig. 3.5.1 plots the resulting timings as a function of the number of edges.

3.5.1.2 Step 2

The second step of the algorithm was tested on two types of graphs, and the running time was measured.

In the first category of tests, graphs are simple chains of various length: each vertex there has a degree of two, except for two boundary vertices, that have a degree of one. The linear behavior of the algorithm's second step can be seen in Fig. 3.5.2.

In the second category of tests (random graphs), graphs have a fixed number of edges, but different branching factors, generated as follows. A fixed number of vertices is created. First, we randomly select a vertex and choose an arbitrary number between one and the desired branching factor (with equal probabilities). This integer determines a number of connections added to this vertex with other untreated vertices of the system (if we have enough of them). These added vertices (adjacent to the initial one) are memorized in an array. Then, we repeat the same procedure for all vertices from this array (choose a number between one and branching factor minus one, create connections, memorize to the new array), and work with the new array. We continue while we have untreated vertices in the system. As can be seen in Fig. 3.5.3, for a given number of edges, running time is inversely proportional to the

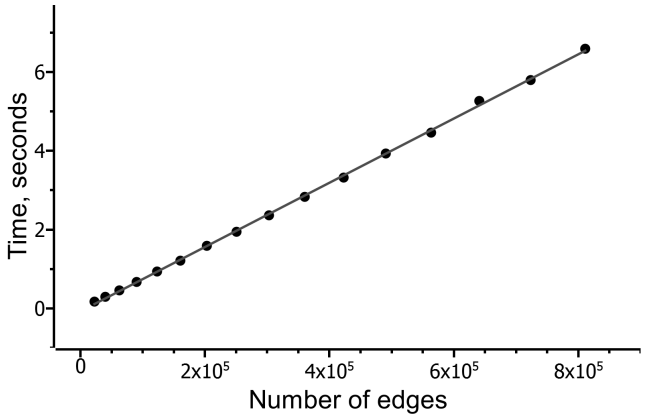


Figure 3.5.2: Computational performance of the algorithm's second step on unbranched chains. Running time as a function of the number of edges in the graph is plotted. The solid line represents a linear fit.

branching factor (solid lines on the figure are fitting curves): for a given number of edges, the largest time corresponds to the unbranched chain tested above.

We also tested the balance-properties of the assembly tree in the following benchmark. For the same branching factor and number of edges in one connected component we generated several random graph configurations and measured the average depth of leaf nodes. Then we varied the number of edges in the system. These results were compared to the the same average for trees obtained by recursive splitting of a connected component in halves (which has $O(N \log N)$ complexity and provides an almost perfectly balanced tree). Two corresponding plots (branching factor is fixed to 4, number of configurations is 50) can be seen in Fig. 3.5.4. They are very similar: maximal difference is 5%.

3.5.1.3 Step 3

The third step of the algorithm was run on systems with different number of points in 3D space. These points are positioned to the nodes of a finite 3D cubical lattice, and are not connected to each other (the random distribution is uniform). The linearity of this step can be seen in Fig. 3.5.5. Results for tests on systems with non-uniformly distributed geometry (biological systems) are presented below.

We also compared on the same benchmark the algorithm proposed here for the third step with two $O(N \log N)$ algorithms: the first algorithm recursively uses the median of median algorithm [202] to split the system along the longest axis so that there is almost the same number of nodes in each subsystem, and the second algorithm splits the system in halves with the help of the nth-element algorithm from the C++ Standard Template Library [182] (in the worst case this splitting can have a $O(N^2)$ complexity but on average it is $O(N)$). These tests were run on Intel Xeon X5450 3GHz processor with 16 GB of RAM (Windows Vista 64-bit

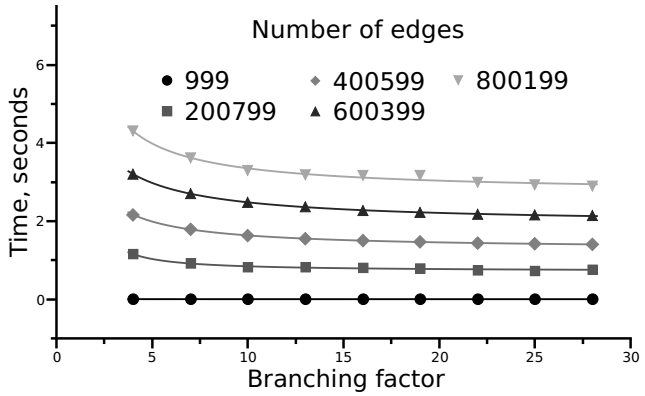


Figure 3.5.3: Running time dependence on the graph branching factor for the second step of the new algorithm. Random graphs with different number of edges are considered and correspond to solid lines (inverse law fits).

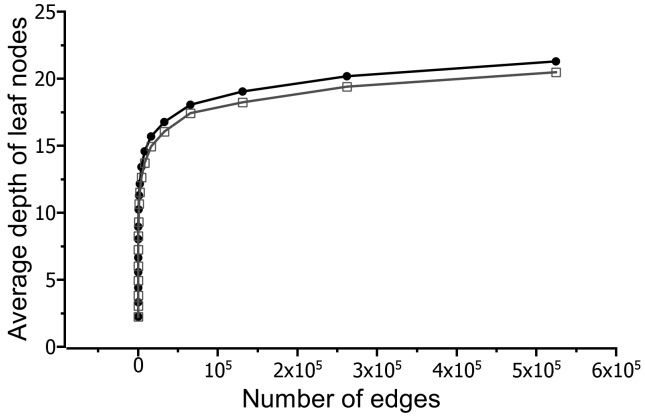


Figure 3.5.4: Average depth of the assembly tree leaf nodes for the second step of the new algorithm, plotted as a function of number of edges in the input graph (random graphs with a fixed branching factor were used). The black line corresponds to our algorithm, the grey line to recursive splitting in halves.

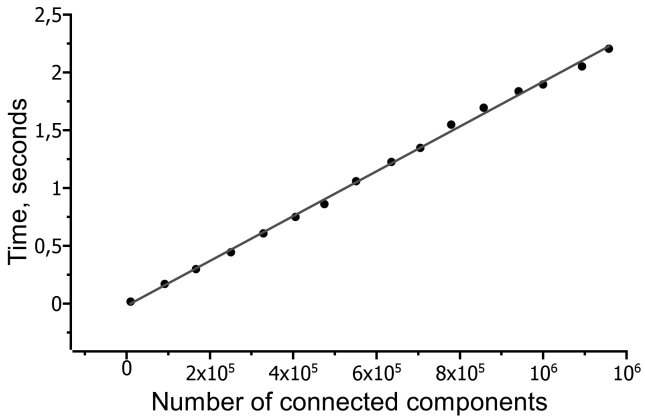


Figure 3.5.5: Computational performance of the algorithm's third step on uniformly distributed systems in 3D. Running time as a function of the number of connected components in the input graph is plotted. The solid line represents a linear fit.

operating system). The results of time comparison are depicted as functions of the number of connected components in Fig. 3.5.6. Our algorithm significantly outperforms both of these algorithms.

The comparison of the average depth of output-tree leaves and its standard deviations as functions of the number of connected components for our algorithm and $O(N \log N)$ algorithms can be found in Fig. 3.5.7.

3.5.2 Molecular graphs

We tested the first and the second steps of algorithm on a subset of the Protein Data Bank files accessible on August, 20-th, 2010. The Protein Data Bank [128] (PDB) is a repository for the 3D structural data of large biological molecules, such as proteins and nucleic acids. For each structure, information is stored in a special PDB file format. In this format, several types of records describe atoms (ATOM, HETATM), bonds (CONECT), chain breaks (TER), and the file (END), store remarks (REMARK), etc. For example, an ATOM record contains the identifier (ID), name, coordinates of the described atom, its occupancy, temperature factor, the chain and the segment that it belongs to. The ATOM record type is used for all atoms in the system, including solvent and ions. CONECT records add important connections inside the structure which can be non-covalent bonds, etc. We provide an example of a simple PDB file in Fig. 3.5.8.

The Protein Data Bank is a key resource in many areas of structural biology, and molecular dynamics simulations are often performed based on structures contained in the PDB. As a result, there is a strong need for fast algorithms which can generate data structures suitable for these simulations, such as assembly trees that we consider in this Chapter.

The subset we chose was the set of files obtained by X-ray crystallography, having a

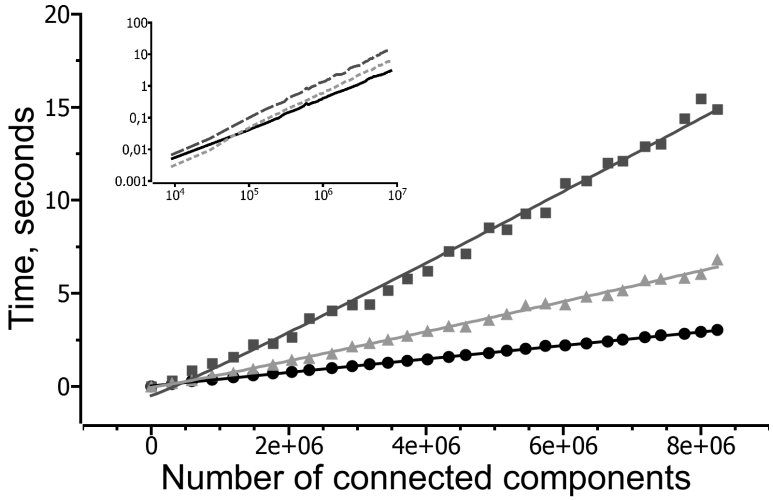


Figure 3.5.6: The third step of the algorithm, comparison with $O(N \log N)$ algorithms on uniformly distributed systems in 3D. Running time for all three algorithms is plotted as a function of number of connected components in the input graph. Lines on the figures are fitting curves, circles stand for our algorithm, triangles for the $O(N \log N)$ algorithm with the nth-element used for splitting, squares for the $O(N \log N)$ algorithm with the median of medians used for splitting. On the inset results are presented in logarithmic scale: the firm line stands for our algorithm, the dotted line for the $O(N \log N)$ algorithm with the nth-element used for splitting, the dashed line for the $O(N \log N)$ algorithm with the median of medians used for splitting.

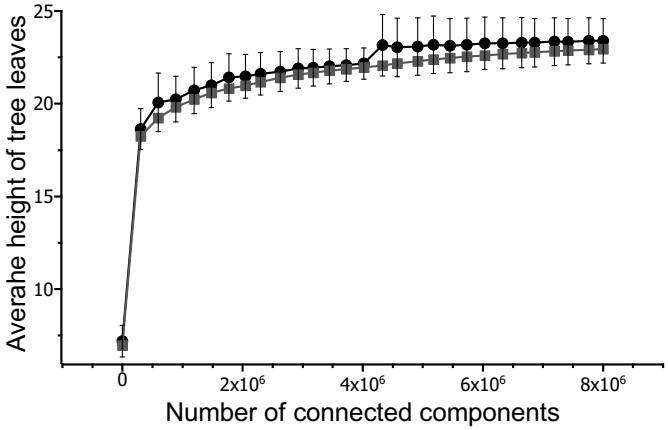


Figure 3.5.7: The third step of the algorithm, comparison with $O(N \log N)$ algorithms on uniformly distributed systems in 3D. Average depth of the assembly tree leaf nodes is plotted as a function of number of connected components in the input graph. Circles stand for our algorithm, squares for both $O(N \log N)$ algorithms (same results for the two versions of splitting techniques).

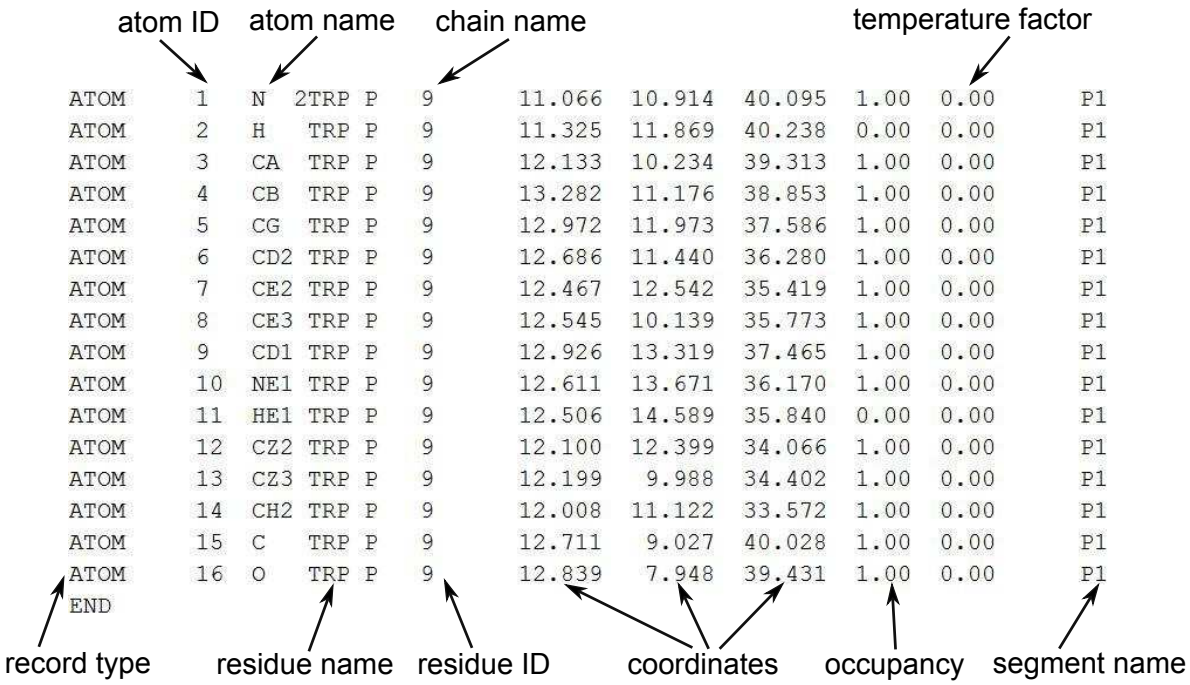


Figure 3.5.8: PDB file, simple example.

resolution better than 2.5 Angstroms, released after the 1-st of January, 1990, and including at least one ATOM record. These files contain less experimental errors for atom positions, which we needed to generate the molecular graphs. The resulting number of files in the benchmark subset was 46137 (out of 67313 files).

In general, cyclic patterns of molecular graphs that should be rigidified depend on the application. For example, in biomolecular structures long cycles from cyclic peptides, cyclic RNA strands, etc. cannot be modeled as rigid bodies. Hence, in order to use our algorithm, we have to locate such cycles and break them. It is not an easy task as, for example, some cycles might be included within larger cycles (as in Fig. 3.1.1), and removing just one edge can be insufficient. Ultimately, we let the user choose which cycles should be broken (depending on the application, because these cycles should not be modeled as rigid bodies), before running our algorithm. Some edges can easily be removed automatically, though, as we describe below.

To have a plausible benchmark here, we use the following way of building molecular graphs from the subset above. We consider atoms as vertices, and bonds as edges. To extract atoms from a file we use ATOM records but not HETATM records. To obtain bonds we do not consider CONECT records in PDB files, but compute covalent bonds based on the distance between two atoms. The criterion for that is $dist < 0.6(r_1 + r_2)$, where $r_1$and r_2 are atom radii (they were taken from the VMD [2] periodic table) and $dist$ is the distance between their centers — this is why we chose to keep only high-resolution structures. We do not add edges

for disulphide bonds, and we do not connect atoms from different molecular chains (different molecules, including ligand-receptor connections), or non-consecutive residues, whatever the distance between them. Alternate locations are not treated neither: if there are several, the one marked with "A" or the one with an empty corresponding field is chosen. This way, we avoid large cycles in our benchmark (the maximum cycle length was 47 for a replication terminator protein in complex with DNA, pdb code 2EFW; an average cycle length was 8.692). We found it acceptable to contract all remaining cycles because small cycles in amino acids and nucleic acids are planar aromatic rings that can be modeled as rigid bodies.

We would like to emphasize that this is a very particular way of retrieving information from the PDB file. However, one is free to choose any other suitable way of doing that (*e.g.* calculate bonds between atoms using covalent radii instead of van der Waals radii). Also here we contract only small cycles in amino acids, but it is possible to contract other subgraphs that may be considered rigid according to the model or application.

For readability of the results, we present the obtained timings as follows. For each step and pass of the algorithm, we first average timings for each distinct number of edges in the molecular graph (on the figures x-axis, if this number is not equal to zero). Then, we average all values in x-intervals of fixed width (for example, 3000 for the second step).

3.5.2.1 Step 1

The timings for the two passes of the first step are shown in Fig. 3.5.9. As expected, the first pass is linear in the number of edges in the graph, and the second pass is linear in the number of cycled edges. On these plots we also show the standard deviation for timings of each interval with the error bars.

3.5.2.2 Step 2

The linear dependence of the running time of the second step of the algorithm on the number of input edges is demonstrated in Fig. 3.5.10. In this case, the algorithm'second step is linear because the branching factor is constant and small for molecular systems. Error bars are again presented on the plot.

3.5.2.3 Step 3

We tested the third step of the algorithm on the full set of the Protein Data Bank files accessible on March, 16-th 2009 (56365 files).

We obtained molecular graphs from the 56365 files by considering atoms as vertices, and covalent bonds as edges, and we ran our algorithm on these graphs. We did not consider CONECT records in PDB files, but we added covalent bonds based on the distance between two atoms. For simplicity in this benchmark evaluating performance only, we rigidified all cycles in molecular graphs and did not break any of them. As the 3D points characterizing connected components, we used their geometric centers.

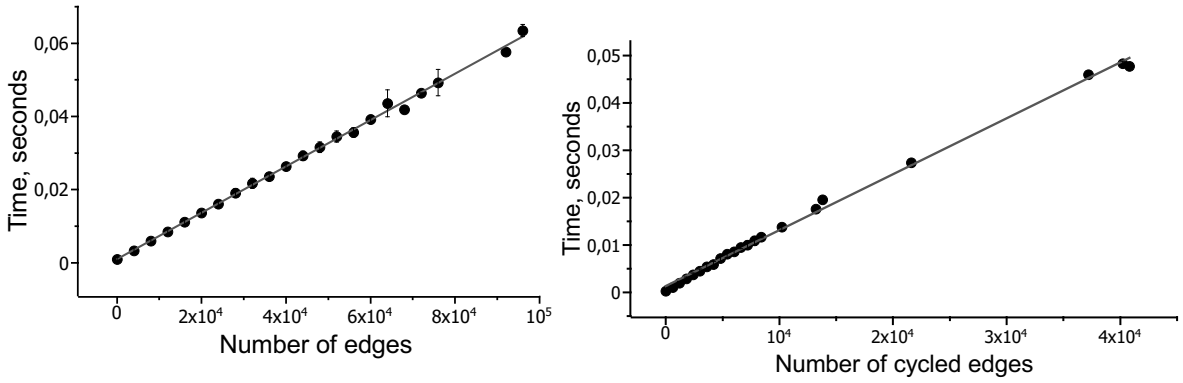


Figure 3.5.9: Running time for passes one and two (left and right figures respectively) of the algorithm's first step on the subset of PDB molecular graphs is plotted as a function of the number of edges in the input graph. Solid lines represent linear fits, error bars indicate the standard deviation for timings in each interval.

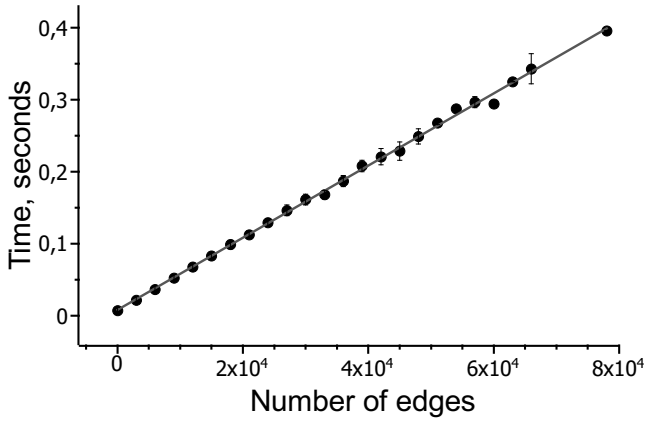


Figure 3.5.10: Running time of step two of the algorithm on the subset of PDB molecular graphs is plotted as a function of the number of edges in the input graph. The solid line represents a linear fit, error bars indicate the standard deviation for timings in each interval.

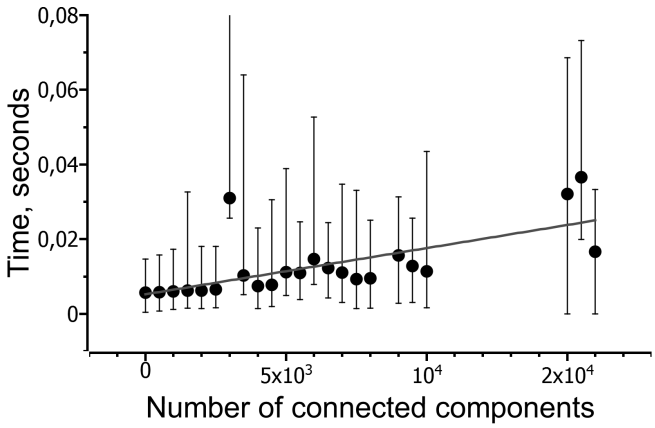


Figure 3.5.11: Running time of the algorithm's step three on the full set of PDB molecular graphs is plotted as a function of the number of edges in the input graph. Solid lines represent linear fits, error bars indicate minimum and maximum times in each interval.

The computational performance of the third step is shown to be a linear function of the number of connected components in Fig. 3.5.11. On the plot for the third step we see a point with a large deviation from the average value. This happens for files with large number of isolated atoms (connected components) which are forming elongated structures in 3D space (in this case, collagen fibers). The time dependence on the number of connected components here is not strictly linear as we have a lot of structures with very different space conformations.

Two examples of assembly trees constructed for molecular graphs are shown in Fig. 3.5.12. The first one (a) is a short poly-alanine with an alpha-helix structure (leaf nodes represent 48 atoms). It illustrates the first and the second steps of the algorithm (tree construction from a single connected component). The second one (b) demonstrates a tree for a small water box (leaf nodes represent 72 water molecules). It is an example of the third step of the algorithm (tree construction from several connected components). Trees for larger molecules are too complex to be visualized on paper, but we feel these two examples are representative.

3.6 Applications of the algorithm

We now give more details on an important application of our algorithm: adaptive torsion-angle molecular quasi-statics. The section concludes with some remarks on other possible uses of assembly trees.

3.6.1 Adaptive torsion-angle molecular quasi-statics

As it was mentioned in the introduction of this Chapter, molecular systems simulated in internal coordinates [40, 41, 42, 43, 44, 45, 46, 47] might need a hierarchical representation. Let us give more details on this application.

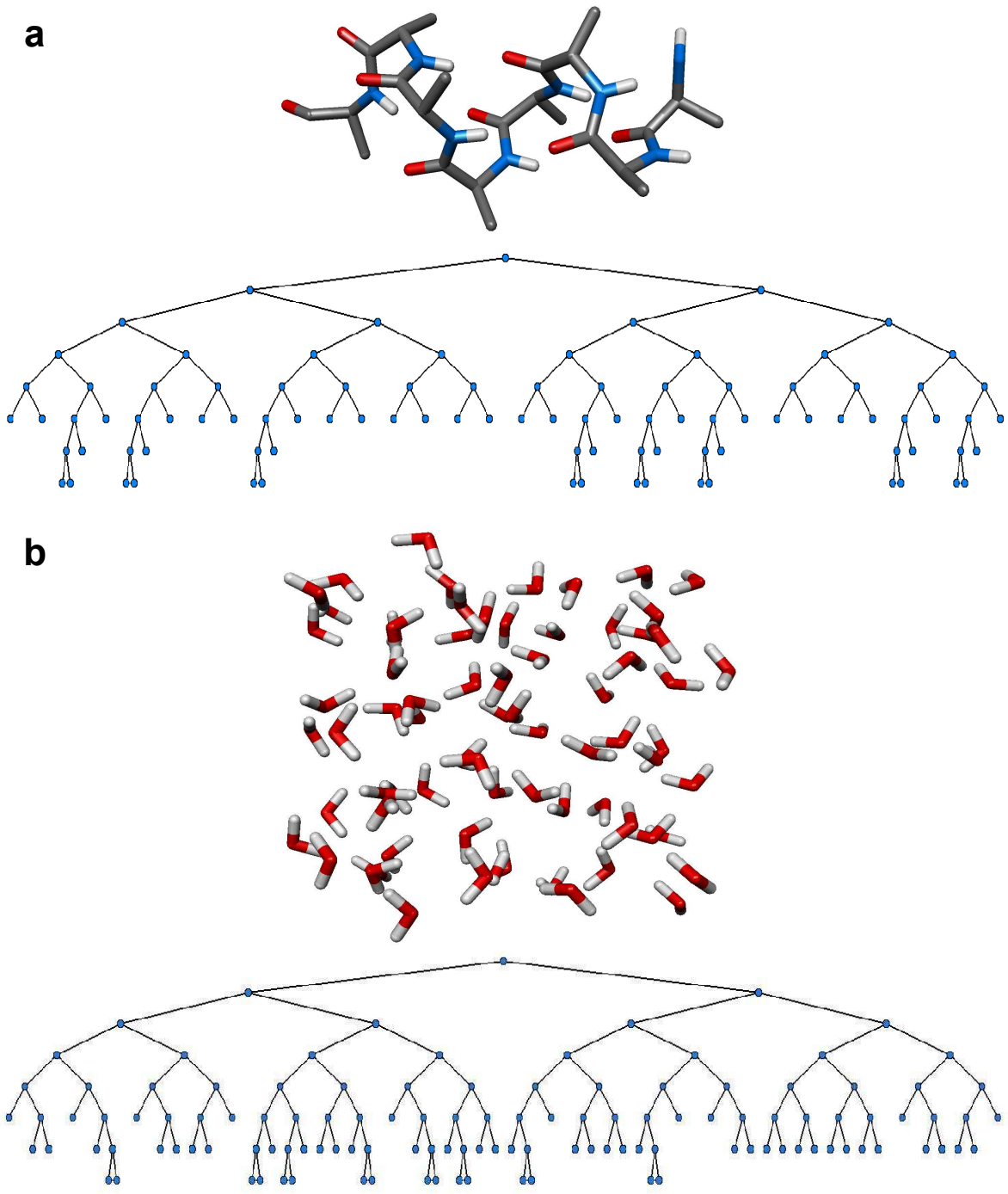


Figure 3.5.12: Examples of assembly trees constructed for molecular graphs: a) the assembly tree corresponding to a single connected component, a helix; b) the assembly tree corresponding to a water box.

Performing a simulation in internal coordinates requires to represent the molecular system as a *kinematic graph*, *i.e.* a graph in which vertices correspond to one or several atoms forming a rigid body, and where graph edges correspond to covalent bonds. Actually, articulated-body representations are also widely used in robotics, biomechanics, modeling and engineering, and a large number of other applications. In general, the vertices of the kinematic graph represent rigid bodies, while its edges represent kinematic constraints between rigid bodies (*e.g.* prismatic constraints, revolute joints, etc.).

The equations of motions in internal coordinates are more involved than their counterparts in Cartesian coordinates. Therefore, it had long been believed that the computational cost of performing a simulation in internal coordinates was approximately cubic in the number of degrees of freedom, due to the cost of inverting the non-constant, dense inertia tensor of the molecular system.

The discovery of recursive algorithms [21, 22, 40], *i.e.* algorithms which rely on a recursive formulation of the equations of motion and scale linearly in the number of degrees of freedom for *acyclic* kinematic graphs, has encouraged simulations of large molecules in internal coordinates. For example, the divide-and-conquer algorithm (DCA) described in Featherstone [21, 22] may be applied to perform molecular modeling and dynamics in internal coordinates [79, 78].

Recently, Redon and Lin [78] and Redon et al. [79] have introduced algorithms for *adaptive* simulation of articulated-body quasi-statics and dynamics. These algorithms make it possible to arbitrarily choose the number of active degrees of freedom at each time step (equivalently, the precision of the simulation), *i.e.* perform a partial update of the system's state at each time step. Precisely, active joints are joints whose position is updated, while rigid joints are frozen: their position is not updated, even if their velocity or acceleration is non-zero. The status of each joint may change at each time step, based on acceleration error metrics [41]. The set of active degrees of freedom (active region) is determined automatically at each time step, based on rigorous error bounds on joint accelerations that can be obtained *a priori*, before computing all joint accelerations. The possibility of finely trading between precision and computational cost may result in significant speedups when a lower number of degrees of freedom is sufficient to describe the motion.

Later on, Rossi et al. [41] have proposed an algorithm for incremental update of molecular energies and forces, which relies on this adaptive articulated-body mechanics algorithm [79]. The result is an adaptive torsion-angle molecular quasi-statics algorithm [41], which enables the user to choose the number of active torsion angles, and in which the computational cost and precision are a function of the number of active degrees of freedom.

The adaptive simulation algorithms [78, 41, 79] rely on Featherstone's DCA [21], and may be seen as a generalization of it: the DCA corresponds to the adaptive algorithm with a zero error tolerance. As a result, both the DCA and the adaptive algorithms need an assembly tree to describe the system's topology. In this case, the assembly tree is a binary tree in which each leaf node represents an individual rigid body, and each internal node represents both the sub-assembly obtained by connecting together the sub-assemblies corresponding to its children,

and the joint (kinematic constraint) used to connect the two children. Therefore, the assembly tree has to be based on the system's topology[9]. In both versions, this decomposition is crucial to perform all computations required to compute the motion (or approximate motion) of the articulated bodies, as the assembly tree characterizes the order in which computations are performed. In particular, computations at the same tree level may be performed in parallel, and the tree should thus be as balanced as possible. Even on a single processor, the balance of the tree is important for the adaptive approach, as a degree of freedom may be activated only if all of its ancestors are active (in other words, the active region has to be a sub-tree of the assembly tree).

Therefore, the algorithm that we provide can be very useful for the DCA-based and adaptive schemes.

Let us give more details on the adaptive torsion-angle quasi-statics algorithm [41].

Precisely, Featherstone [21, 22] shows that the dynamics of an articulated body can be described by the following articulated-body equation:

$$\mathbf{a} = \mathbf{\Phi}\mathbf{f} + \mathbf{b}, \tag{3.6.1}$$

where $\mathbf{a}$ is the composite acceleration of the articulated body, $\mathbf{\Phi}$ is the composite inverse inertia of the articulated body, $\mathbf{f}$ is a composite kinematic constraint force and $\mathbf{b}$ is a composite bias acceleration, due to external forces and torques. Then, assuming C denotes an articulated body formed by assembling two articulated bodies A and B after computing the coefficients of the leaf nodes (the rigid bodies) we can recursively (bottom-up) obtain the inverse inertia and bias acceleration of C from those of A and B:

$$\mathbf{\Phi}^C = \mathbf{\Phi}^C\left(\mathbf{\Phi}^A, \mathbf{\Phi}^B\right), \quad \mathbf{b}^C = \mathbf{b}^C\left(\mathbf{b}^A, \mathbf{b}^B, \mathbf{Q}^C\right), \tag{3.6.2}$$

where $\mathbf{Q}^C$ is a torque applied to the joint connecting A and B. Then (top-down), the kinematic constraint forces $\mathbf{f}^A$and $\mathbf{f}^B$ and the acceleration $\ddot{\mathbf{q}}^C$ of the joint connecting A and B can be calculated:

$$\left(\mathbf{f}^A, \mathbf{f}^B\right) = g\left(\mathbf{f}^C\right), \quad \ddot{\mathbf{q}}^C = \ddot{\mathbf{q}}^C\left(\mathbf{\Phi}^A, \mathbf{\Phi}^B, \mathbf{f}^C\right), \tag{3.6.3}$$

Redon and Lin [78] have shown that it is possible to determine *a priori* the subset of largest joint accelerations, *before* computing all of them, which makes it possible to perform a partial state update at each time step, and finely trade between precision and computational cost. Precisely, the set of important accelerations is determined automatically based on an *acceleration metric* (weighted sum of the joint accelerations in an articulated body C):

$$A(C) = \sum \ddot{\mathbf{q}}_i^T \mathbf{A}_i \ddot{\mathbf{q}}_i, \tag{3.6.4}$$

which can be computed in constant time through

[9] However, *spatial* decomposition (*i.e.* grouping vertices based on spatial proximity) could be useful for some other applications.

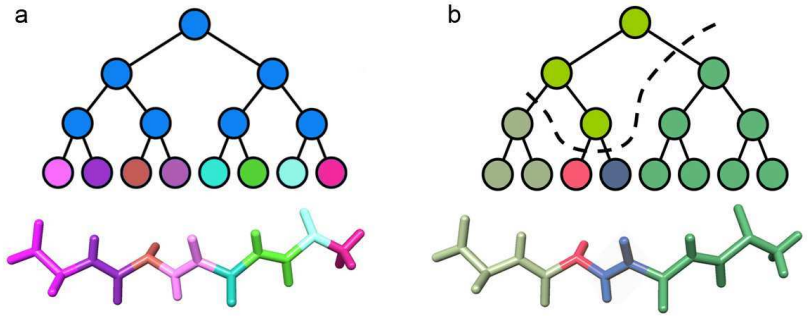


Figure 3.6.1: Assembly tree of a short polypeptide. The left part (a) shows the assembly tree of a tetra-alanine (CHARMM19 model). Colors indicate the correspondence between leaf nodes and rigid bodies. The right part (b) shows a possible active region (green nodes) produced by the adaptive algorithm during the simulation, when the user has allowed for three simultaneously active joints only. This results in a partitioning of the molecule into four rigid bodies (one color per rigid or rigidified body — colors also indicate the correspondence between rigid bodies and leaf or internal nodes).

$$A(C) = \left(\mathbf{f}^C\right)^T \mathbf{\Psi}^C \mathbf{f}^C + \left(\mathbf{f}^C\right)^T \mathbf{p}^C + \eta^C, \tag{3.6.5}$$

since the coefficients $\mathbf{\Psi}^C, \mathbf{p}^C, \eta^C$ can be recursively computed bottom-up, following the tree structure [78].

The set of important accelerations is determined top-down and, as a result, the active region (the set of simulated joints) is a sub-tree of the assembly tree (Fig. 3.6.1). The active region is updated at each time step, based on a user-defined error tolerance on joint accelerations, or a user-defined maximum number of active joints [78].

Two examples of the results of the adaptive torsion-angle quasi-statics algorithm are shown in Fig. 3.6.2 and 3.6.3. Rigid body clusters are represented by different colors. When a force is applied or the conformation of molecules change, the distribution of joint accelerations is modified, and a different clustering is automatically determined by the adaptive algorithm.

In Fig. 3.6.2, on the left, a force is interactively applied by the user to the middle of a poly-alanine, to modify its structure. The deformation occurs mostly in the neighborhood of the point of application, and the degrees of freedom are activated around this point. To maintain constant the total number of degrees of freedom in the molecule, some degrees of freedom in the more stable ends of the molecule are frozen (this is made visible by the coloring update). The same procedure is repeated for a bigger molecule (chain A of the complex with a PDB code 1AC1) on the right figure.

In Fig. 3.6.3, a ditryptophan ligand is interactively docked to a monomer of a KcsA potassium channel by the user. Before docking, the nine active degrees of freedom are homogeneously distributed within the monomer, since the structure is stable. During docking,

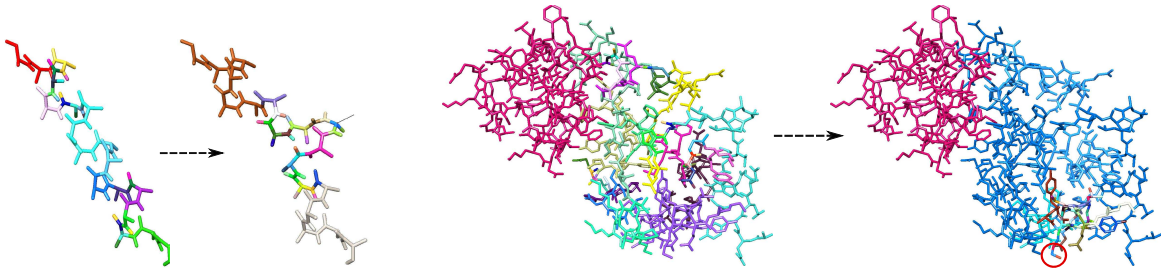


Figure 3.6.2: Adaptivity example, the force is applied to a poly-alanine (left) and to a part of a protein complex (PDB code 1AC1, right). Colors represent rigid body clusters (hence show the location of active torsion angles). For the right molecule the point where the force is applied is marked with a red circle.

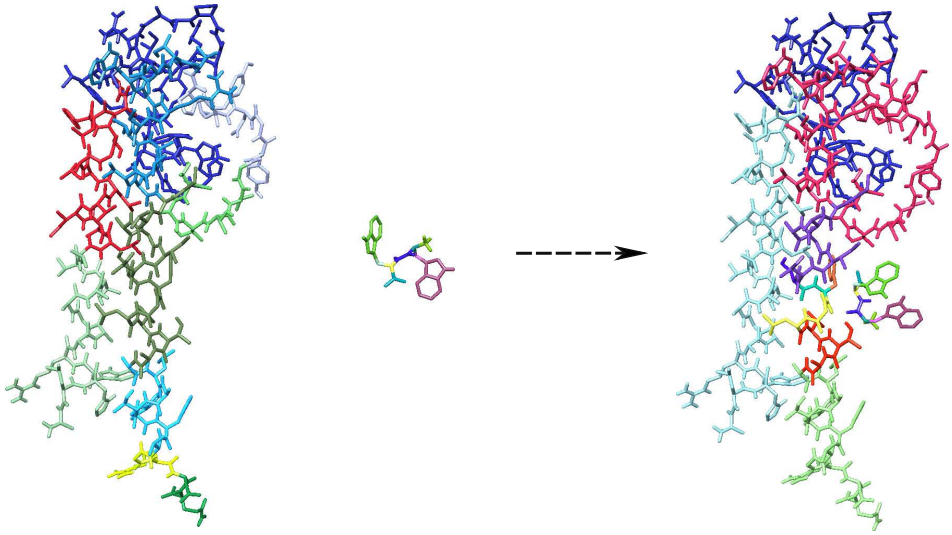


Figure 3.6.3: Interactive docking example. Degrees of freedom are represented by different colors.

however, the interaction between the ligand and the monomer automatically produces a redistribution of the active degrees of freedom around the docking site, to better approximate the interaction. In this case as well, the small number of active degrees of freedom allows us to efficiently perform interactive flexible docking of the ligand into the protein.

3.6.2 Other applications

We would like to stress that our tree construction algorithm can be useful for any application relying on the DCA algorithm or adaptive algorithms, or even other types of hierarchy-based torsion-angle dynamics algorithms [203].

It has been also shown elsewhere that it can be employed in other applications, *e.g.* minimization of symmetrical systems in internal coordinates [76].

3.7 Conclusion

We have presented a fast algorithm for building an assembly tree associated to a molecular graph, which may contain cycles and multiple connected components. This data structure can be useful in a number of applications using internal coordinates, including those performing adaptive computations on such graphs [79, 78, 41]. More generally, we believe that our algorithm might be useful for other algorithms (*e.g.* parallel ones) that rely on hierarchical representations of molecular graphs.

Our algorithm consists of three main steps: (1) contracting cycles in the input graph; (2) building an assembly tree for the acyclic connected graph; (3) building an assembly tree of the complete graph comprising several connected components. We have studied the complexity of each step of the algorithm both theoretically and experimentally on several sets of benchmarks. The first step of our algorithm is linear in the number of edges in the input graph. The second one is linear in the number of edges in the contracted graph, but also depends on the branching factor of this graph. We also studied the balance properties of the obtained tree. The third step is linear in the number of connected components for uniformly distributed systems and is dependent on the system's geometry in the general case. There are some limitations for the algorithm proposed. For example, the choice of the auxiliary parameter (the number of cells along each axis at the third step) can influence the result.

We have demonstrated applications of our algorithm to adaptive torsion-angle quasi-statics.

In the future, we would like to study possible extensions, as well as other applications of our algorithm. For example, it would be interesting to investigate the procedure of dynamic tree updates after inserting or deleting elements from the graph, or when changing the graph topology or geometry.

Chapter 4

ARPS: Adaptively Restrained Particle Simulations

Les potentiels d'interaction utilisés dans les simulations de particules sont généralement écrits comme une somme de termes qui dépendent seulement de quelques positions relatives de particules. Les méthodes de simulation traditionnelles déplacent toutes les particules à chaque pas de temps, et peuvent donc dépenser beaucoup de temps pour mettre à jour toutes les forces entre particules.

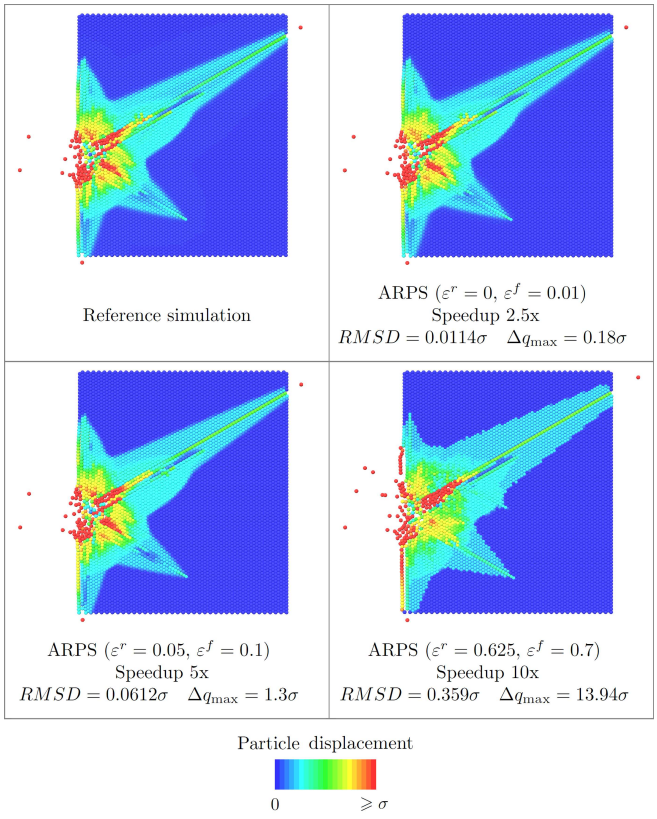
Dans le Chapitre précédent, nous avons parlé des algorithmes adaptatifs qui simplifient les simulations en réduisant le nombre de degrés de liberté dans le système. Ces algorithmes utilisent des méthodes incrémentales pour mettre à jour les forces entre particules et, par conséquent, peuvent accélérer les simulations moléculaires. Dans ces méthodes, cependant, l'énergie du système n'est pas conservée, et aucune correction théorique n'a été fournie pour résoudre ce problème. Aucune étude sur les propriétés du système n'a été non plus réalisée.

C'est pour cela que, dans ce Chapitre, nous introduisons une méthode générale et mathématiquement fondée que nous appelons les Simulations de particules restreintes de façon adaptative (ARPS, Adaptively Restrained Particle Simulations). Cette méthode accélére les simulations de particules en activant et désactivant les degrés de liberté en positions du système, tout en laissant les moments évoluer. Nous illustrons ARPS sur plusieurs expériences numériques, notamment (a) un exemple de cascade de collisions qui montre comment ARPS permet de choisir finement le compromis entre précision et vitesse, et (b) une étude de polymère solvaté qui montre comment il est possible de calculer rapidement des proprietés statiques d'équilibre correctes avec ARPS.

Interaction potentials used in particle simulations are typically written as a sum of terms that depend on just a few relative particle positions. Traditional simulation methods move all particles at each time step, and may thus spend a lot of time updating inter-particle forces.

In the previous Chapter, we have discussed adaptive algorithms that simplify simulations by reducing the number of degrees of freedom in the system. These algorithms use incremental methods to update inter-particle forces and, thus, may accelerate molecular simulations. In these adaptive algorithms, however, the energy of the system is not conserved, and no theoretical correction was provided to solve this problem. No further study on the system's properties was conducted either.

That is why, in this Chapter, we introduce a general, theoretically-grounded method to speed up particle simulations that we call ARPS: Adaptively Restrained Particle Simulations. This method adaptively switches on and off positional degrees of freedom, while letting momenta evolve. We illustrate ARPS on several numerical experiments, including (a) a collision cascade example that demonstrates how ARPS make it possible to smoothly trade between precision and speed, and (b) a polymer-in-solvent study that shows how one may efficiently compute static equilibrium properties with ARPS.



4.1 Introduction

In the previous Chapter, as an application example, we have discussed adaptive algorithms that simplify simulations by reducing the number of degrees of freedom in the system. These algorithms use incremental methods to update inter-particle forces and, thus, may accelerate molecular simulations. In these adaptive algorithms, however, the energy of the system is not conserved, and no theoretical correction was provided to solve this problem. No further study on the system's properties was conducted either.

That is why we were interested in developing a general, theoretically-grounded method that would accelerate simulations but would also sample the phase space correctly and allow to collect exact statistical averages. As molecular systems are usually simulated through writing a Hamiltonian function and deriving equations of motion from it, we have chosen Hamiltonian dynamics for our approach. As a result, it can be applied to a broader type of simulations: particle simulations.

Particle simulations are widely used not only in molecular dynamics, but also in fluid dynamics (dynamics of liquid, gas and plasma), celestial mechanics [204, 205], and even in computer graphics [206, 207]. In all of these fields, faster simulations may result in progress on many challenging problems, *e.g.* protein folding [17, 18], molecular docking [5, 131, 132], molecular solvation [19], diffusion across biomembranes [20], fracture in metals[208], ion implantation [209], etc.

Novel, general approach to speed up particle simulations that we introduce in this Chapter is called Adaptively Restrained Particle Simulations (ARPS). Our approach relies on an adaptively restrained (AR) Hamiltonian, and works by adaptively switching positional degrees of freedom on and off repeatedly during a simulation, while letting momenta evolve. This is achieved by making the inverse inertia of the particles a general function of phase-space coordinates. Simulating the adaptively restrained Hamiltonian produces approximate trajectories of the original system that depend on user-defined error thresholds. The main advantages of our approach are that (a) it is mathematically grounded and is able to produce long, stable simulations; (b) it does not require modifications to the simulated interaction potential, so that any suitable existing force-field can be directly used with ARPS; (c) as the same potential is used for the whole system, there is no resolution change, and, therefore, the reverse-mapping problem (an important problem for many coarse-graining methods) does not arise (d) under frequently-used assumptions on the interaction potential, ARPS make it possible to reduce the number of forces that have to be updated at each time step, which may significantly speed up simulations; (e) ARPS can be combined with numerous existing methods, such as fast algorithms to compute long-range interactions [74, 75] (although incremental versions may have to be designed), as well as techniques aimed at accelerating sampling [97, 85, 98]; (f) when performing constant-energy simulations, ARPS allow users to finely and continuously trade between precision and computational cost, and rapidly obtain approximate trajectories; (g) most important, when performing Adaptively Restrained Molecular Dynamics (ARMD) in the canonical (NVT) ensemble, correct static equilibrium properties can be computed.

In this Chapter we propose one specific choice of the inverse inertia matrix for simulations

in Cartesian coordinates. In this case, positional degrees of freedom of individual particles are switched on and off during the simulation, and the inverse inertia matrix is diagonal. Moreover, the simplification threshold – hence the trade-off between precision and cost – may be chosen for each particle independently, so that users may arbitrarily focus ARPS on specific regions of the simulated system (*e.g.* a polymer in a solvent).

It is interesting to note that, in our approach, we may not specify the region of interest in the simulation: the method will automatically determine the most relevant simulation parts (it is demonstrated below in one of the examples: collision cascade study). However, with an appropriate choice of the inverse inertia matrix it might be possible to impose a specific region in space to be active all the time during the simulation. We do not fix either a level of coarse-graining in the system, to *e.g.* water molecules. Depending on the conditions, larger blocks may move together.

The rest of this Chapter is organized as follows. We first present the theoretical basis of ARPS approach and discuss its properties. Then, we provide efficient integration algorithms to perform AR simulations. Finally, ARPS is illustrated on several numerical experiments, including (a) a collision cascade example that demonstrates how ARPS make it possible to smoothly trade between precision and speed, and (b) a polymer-in-solvent study that shows how one may efficiently compute static equilibrium properties with ARPS.

4.2 Theory

Let us now describe the theoretical aspects of our novel approach, ARPS.

4.2.1 General Hamiltonian

A classical way of studying a particle system is writing a Hamiltonian function and deriving equations of motion from it. A Hamiltonian is usually expressed as a sum of kinetic and potential energies of the system: $H = K + V$. The precise form of these energies, however, is chosen according to the system under study.

Let $H(\mathbf{q}, \mathbf{p})$ denote the Hamiltonian commonly associated to a system of N particles in 3D [23]:

$$H(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p} + V(\mathbf{q}), \tag{4.2.1}$$

where a pair $(\mathbf{q}, \mathbf{p})$ is a vector of $6N$ phase-space coordinates, vector $\mathbf{q}$ represents generalized coordinates and $\mathbf{p}$ – generalized momenta, $\mathbf{M}$ is a $3N \times 3N$ diagonal mass matrix (which might depend on positions $\mathbf{q}$), and $V(\mathbf{q})$ is an interaction potential.

For example, in Cartesian coordinates, vector $\mathbf{q}$ represents positions of all particles, and $\mathbf{p}$ – their momenta:

$$\mathbf{q} = \begin{bmatrix} q_{1,x} \\ q_{1,y} \\ q_{1.z} \\ \vdots \\ q_{N,x} \\ q_{N,y} \\ q_{N,z} \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} p_{1,x} \\ p_{1,y} \\ p_{1.z} \\ \vdots \\ p_{N,x} \\ p_{N,y} \\ p_{N,z} \end{bmatrix}.$$

The first term of the sum in equation (4.2.1) is the most common form of the kinetic energy for the system of N particles.

The second term, the interaction potential, is assumed to depend on positions of the particles but not on their momenta. Typically, this potential (derived from classical or quantum computations) may be written as a sum of terms that depend on a few relative positional degrees of freedom only [33, 50, 35]. Different potentials are adapted to different types of atoms or interactions [33, 50, 35].

The equations of motion in this case take the form:

$$\begin{aligned} \dot{\mathbf{p}} &= -\frac{\partial H}{\partial \mathbf{q}} = -\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}}, \\ \dot{\mathbf{q}} &= \frac{\partial H}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p}, \end{aligned} \tag{4.2.2}$$

where $\dot{\mathbf{q}}$ and $\dot{\mathbf{p}}$ are time derivatives of particle positions and momenta, respectively.

In our approach, we introduce an *Adaptively Restrained (AR) Hamiltonian*, *i.e.* a Hamiltonian with a modified inverse inertia matrix $\Phi(\mathbf{q}, \mathbf{p})$:

$$H_{AR}(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T \Phi(\mathbf{q}, \mathbf{p})\mathbf{p} + V(\mathbf{q}). \tag{4.2.3}$$

We will show how, thanks to the matrix $\Phi(\mathbf{q}, \mathbf{p})$, we will be able to switch positional degrees of freedom on and off, with the help of two thresholds involved in the definition of $\Phi(\mathbf{q}, \mathbf{p})$. We note, however, that the potential $V(\mathbf{q})$ does not need to be modified, so that any suitable existing force-field can be directly used in ARPS.

The *adaptive* equations of motion may now be deduced from the AR Hamiltonian:

$$\begin{aligned} \dot{\mathbf{p}} &= -\frac{\partial H_{AR}}{\partial \mathbf{q}} = -\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}} - \frac{1}{2}\mathbf{p}^T \frac{\partial \Phi(\mathbf{q}, \mathbf{p})}{\partial \mathbf{q}}\mathbf{p}, \\ \dot{\mathbf{q}} &= \frac{\partial H_{AR}}{\partial \mathbf{p}} = \Phi(\mathbf{q}, \mathbf{p})\mathbf{p} + \frac{1}{2}\mathbf{p}^T \frac{\partial \Phi(\mathbf{q}, \mathbf{p})}{\partial \mathbf{p}}\mathbf{p}. \end{aligned} \tag{4.2.4}$$

4.2.2 Statistics

We now specify our study for the case of molecular dynamics simulations. Our approach in this case is called Adaptively Restrained Molecular Dynamics (ARMD).

Statistical properties of a molecular system are studied in a statistical ensemble. In this work, we will deal with two principal ensembles: microcanonical ensemble (NVE) and canonical ensemble (NVT) [23].

The first one, NVE, is directly simulated by the Hamiltonian dynamics. This ensemble describes the thermodynamical properties of an isolated system, and, thus, preserves the energy of the system. In other words, it samples only one energy isoline in the phase space, and this isoline is specified by the initial conditions. The second ensemble, NVT, passes through all energy levels sampling a canonical measure. It has a well-defined temperature T. One way to perform an NVT simulation is through Langevin dynamics. Langevin dynamics is a stochastic dynamics that models a situation where a Hamiltonian system is coupled to a thermostat.

To simulate the NVT ensemble we perform an *adaptive Langevin dynamics simulation, i.e.* a Langevin dynamics simulation of the adaptively restrained Hamiltonian.

To do so, we consider the general form of Langevin equations [107]:

$$\begin{aligned} dq_t &= \nabla_p H_{AR}(q_t, p_t)dt, \\ dp_t &= -\nabla_q H_{AR}(q_t, p_t)dt - \gamma \nabla_p H_{AR}(q_t, p_t)dt + \sigma dW_t, \end{aligned} \tag{4.2.5}$$

where $t \to dW_t$ is a $3N$-dimensional standard Brownian motion, and σ and γ are $3N \times 3N$ real matrices. The following fluctuation-dissipation relation should also be satisfied: $\sigma\sigma^T = 2\gamma/\beta$ for $\beta = 1/k_B T$, where k_B is a Boltzmann constant.

We now show how static equilibrium properties can be determined using AR simulations in the NVT ensemble.

Due to the fact that we have an underlying Hamiltonian to perform AR simulations, we can recover the original statistics by computations similar to those performed in importance sampling [23]. Indeed, we can consider that the AR Hamiltonian (H_{AR}) is a biased version of the original one (H):

$$\begin{aligned} H_{AR} &= \frac{1}{2}\mathbf{p}^T \Phi(\mathbf{q}, \mathbf{p})\mathbf{p} + V(\mathbf{q}) = \\ &= \frac{1}{2}\mathbf{p}^T \mathbf{M}^{-1}\mathbf{p} + V(\mathbf{q}) + \frac{1}{2}\mathbf{p}^T(\Phi(\mathbf{q}, \mathbf{p}) - \mathbf{M}^{-1})\mathbf{p} = \\ &= H + \frac{1}{2}\mathbf{p}^T(\Phi(\mathbf{q}, \mathbf{p}) - \mathbf{M}^{-1})\mathbf{p}, \\ \\ H_{AR} &= H + V_{AR}(\mathbf{q}, \mathbf{p}). \end{aligned} \tag{4.2.6}$$

Thus, to compute an average $\langle \mathbf{A} \rangle_H$:

$$\langle \mathbf{A} \rangle_H = \frac{\int \mathbf{A}(\mathbf{q}, \mathbf{p}) e^{-\frac{H(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q} d\mathbf{p}}{\int e^{-\frac{H(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q} d\mathbf{p}}, \tag{4.2.7}$$

one may write the following:

$$\langle \mathbf{A} \rangle_H = \frac{\mathbf{A}(\mathbf{q},\mathbf{p})\, e^{-\frac{H_{AR}(\mathbf{q},\mathbf{p})-V_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}}{\int e^{-\frac{H_{AR}(\mathbf{q},\mathbf{p})-V_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}} = \\ = \frac{\int \mathbf{A}(\mathbf{q},\mathbf{p}) e^{\frac{V_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} e^{-\frac{H_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}}{\int e^{\frac{V_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} e^{-\frac{H_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}} = \frac{\langle \mathbf{A} e^{\frac{V_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} \rangle_{H_{AR}}}{\langle e^{\frac{V_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} \rangle_{H_{AR}}}. \tag{4.2.8}$$

Therefore, phase-space samples generated according to the AR distribution are then weighted (remark how the weight of a phase space sample is 1 if the system is completely active, and smaller than 1 elsewhere).

When the variable of interest $\mathbf{A}$ *only depends on positions and the Hamiltonian is separable* (*e.g.* the inverse mass matrix depends only on momenta) we can get the correct static equilibrium property straight away (consider that $H(\mathbf{q},\mathbf{p}) = K(\mathbf{p}) + V(\mathbf{q})$):

$$\langle \mathbf{A}(\mathbf{q}) \rangle_{H_{AR}} = \frac{\int \mathbf{A}(\mathbf{q}) e^{-\frac{H_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}}{\int e^{-\frac{H_{AR}(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}} = \frac{\int \mathbf{A}(\mathbf{q}) e^{-\frac{H(\mathbf{q},\mathbf{p})+V_{AR}(\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}}{\int e^{-\frac{H(\mathbf{q},\mathbf{p})+V_{AR}(\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}} = \\ = \frac{\int \mathbf{A}(\mathbf{q}) e^{-\frac{V(\mathbf{q})}{k_B T}} d\mathbf{q} \int e^{-\frac{K(\mathbf{p})+V_{AR}(\mathbf{p})}{k_B T}} d\mathbf{p}}{\int e^{-\frac{V(\mathbf{q})}{k_B T}} d\mathbf{q} \int e^{-\frac{K(\mathbf{p})+V_{AR}(\mathbf{p})}{k_B T}} d\mathbf{p}} = \frac{\int \mathbf{A}(\mathbf{q}) e^{-\frac{V(\mathbf{q})}{k_B T}} d\mathbf{q}}{\int e^{-\frac{V(\mathbf{q})}{k_B T}} d\mathbf{q}} = \\ = \frac{\int \mathbf{A}(\mathbf{q}) e^{-\frac{V(\mathbf{q})}{k_B T}} d\mathbf{q} \int e^{-\frac{K(\mathbf{p})}{k_B T}} d\mathbf{p}}{\int e^{-\frac{V(\mathbf{q})}{k_B T}} d\mathbf{q} \int e^{-\frac{K(\mathbf{p})}{k_B T}} d\mathbf{p}} = \frac{\int \mathbf{A}(\mathbf{q}) e^{-\frac{H(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}}{\int e^{-\frac{H(\mathbf{q},\mathbf{p})}{k_B T}} d\mathbf{q}d\mathbf{p}} = \langle \mathbf{A}(\mathbf{q}) \rangle_H. \tag{4.2.9}$$

It is interesting to note that due to this representation of the Hamiltonian (4.2.1), one can consider $V_{AR}(\mathbf{q},\mathbf{p}) = 0.5\,\mathbf{p}^T(\Phi(\mathbf{q},\mathbf{p}) - \mathbf{M}^{-1})\mathbf{p}$ as a modified potential, and not a modified kinetic energy. Such potentials depending on particles momenta have been proposed before, *e.g.* the Pauli potential [210].

4.2.3 Integration

To perform particle simulations, the equations of motion should be numerically integrated in time [87]. We will now discuss a choice of an integration scheme for the AR simulations.

4.2.3.1 Integration in the NVE ensemble

For the microcanonical ensemble (NVE) simulation, we search for a symplectic integrator: it is well-suited for long-term integrations of Hamiltonian systems, as it preserves the geometric

structure of the Hamiltonian flow (the flow ϕ_t of a Hamiltonian system is the mapping that advances the solution by time t). Let us explain what the symplectic integrator means.

Symplectic integrators A linear mapping X is symplectic if

$$X^T J X = J \quad \text{with } J = \begin{pmatrix} 0 & I \\ -I & 0 \end{pmatrix},$$

where I is an identity matrix.

In symplectic integrators every one-step mapping is symplectic.

As a result, symplectic integrators preserve volume in the phase space and keep the fluctuations of energy bounded. The produced phase-space trajectories do not cross. A symplectic integrator conserves the value of a modified, or shadow, Hamiltonian much better than the energy of the system (which might drift after a sufficient simulation time). Shadow Hamiltonian is defined by th asymptotic expansion in powers of the integration step size [211].

Several symplectic integrators are compared in Ref. [212] and the progress on recent efficient integrators can be found in Ref. [213].

Partitioned Euler method In our implementation, we use a so-called *partitioned* Euler method, which for partitioned systems:

$$\begin{aligned} \dot{v} &= b(u, v), \\ \dot{u} &= a(u, v), \end{aligned}$$

results in the following set of equations for a numerical integrator:

$$\begin{aligned} v_{n+1} &= v_n + b(u_n, v_{n+1})h, \\ u_{n+1} &= u_n + a(u_n, v_{n+1})h, \end{aligned} \tag{4.2.10}$$

where the lower indices stand for time step numbers and h is a time step size. This method is symplectic [87] and is also called symplectic Euler method.

For this integrator, equations (4.2.4) become:

$$\begin{aligned} p_{n+1} &= p_n + \left(-\frac{\partial V(q_n)}{\partial q_n} - \frac{1}{2} p_{n+1}^T \frac{\partial \Phi(q_n, p_{n+1})}{\partial q_n} p_{n+1} \right) h, \\ q_{n+1} &= q_n + \left(\Phi(q_n, p_{n+1}) p_{n+1} + \frac{1}{2} p_{n+1}^T \frac{\partial \Phi(q_n, p_{n+1})}{\partial p_{n+1}} p_{n+1} \right) h. \end{aligned} \tag{4.2.11}$$

If the Hamiltonian is separable (*i.e.* when inverse mass matrix depends only on momenta), a lot of simple integrators become symplectic *and* explicit (*i.e.* p_{n+1} does not appear on the right side of the first equation anymore). Precisely, the partitioned Euler method in this case takes the form:

$$
\begin{aligned}
p_{n+1} &= p_n + \left(-\frac{\partial V(q_n)}{\partial q_n}\right) h, \\
q_{n+1} &= q_n + \left(\Phi(p_{n+1}) p_{n+1} + \frac{1}{2} p_{n+1}^T \frac{\partial \Phi(p_{n+1})}{\partial p_{n+1}} p_{n+1}\right) h.
\end{aligned}
\tag{4.2.12}
$$

4.2.3.2 Integration in the NVT ensemble

For an NVT simulation, one possibility is to use the following splitting scheme [107]: a half step for the Langevin part of the equations, a full time step for the Hamiltonian part, and again a half step for the Langevin part:

$$
\begin{aligned}
p_{n+1/2} &= p_n - \left(\frac{\partial H_{AR}(q_n, p_{n+1/2})}{\partial q_n} + \gamma \frac{\partial H_{AR}(q_n, p_{n+1/2})}{\partial p_{n+1/2}}\right) \frac{h}{2} + \sigma G_n \sqrt{\frac{h}{2}}, \\
q_{n+1} &= q_n + \left(\frac{\partial H_{AR}(q_n, p_{n+1/2})}{\partial p_{n+1/2}}\right) h, \\
p_{n+1} &= p_{n+1/2} - \left(\frac{\partial H_{AR}(q_{n+1}, p_{n+1})}{\partial q_{n+1}} + \gamma \frac{\partial H_{AR}(q_{n+1}, p_{n+1})}{\partial p_{n+1}}\right) \frac{h}{2} + \sigma G_{n+1/2} \sqrt{\frac{h}{2}},
\end{aligned}
\tag{4.2.13}
$$

where $\{G_k\}$ is a sequence of i.i.d. Gaussian random vectors with zero mean and the identity covariance matrix. In this case, two equations of three in (4.2.13) are implicit. To solve an implicit equation $p_{n+1} = g(p_{n+1})$, one may use a simple fixed-point algorithm: $p_{n+1}^{k+1} = g\left(p_{n+1}^k\right)$, where the upper index k denotes the iteration number of this algorithm. For the last equation in eq. (4.2.13) the fixed-point algorithm looks as follows:

$$
p_{n+1}^{k+1} = p_{n+1/2} - \left(\frac{\partial H_{AR}(q_{n+1}, p_{n+1}^k)}{\partial q_{n+1}} + \gamma \frac{\partial H_{AR}(q_{n+1}, p_{n+1}^k)}{\partial p_{n+1}^k}\right) \frac{h}{2} + \sigma G_{n+1/2} \sqrt{\frac{h}{2}}. \tag{4.2.14}
$$

In practice, we found that about 3 iterations were sufficient for the method to converge.

4.2.4 Choosing the inverse mass matrix

We will now describe how the matrix $\Phi(\mathbf{q}, \mathbf{p})$ may be chosen to perform AR simulations. This matrix must specify *how* and *when* positional degrees of freedom are switched on and off. We now propose such a matrix for particle simulations in Cartesian coordinates.

4.2.4.1 How to switch

We choose to switch positional degrees of freedom on and off for each particle independently, and, to do so, we use a diagonal matrix $\Phi(\mathbf{q}, \mathbf{p})$. Each diagonal 3×3 block of this matrix

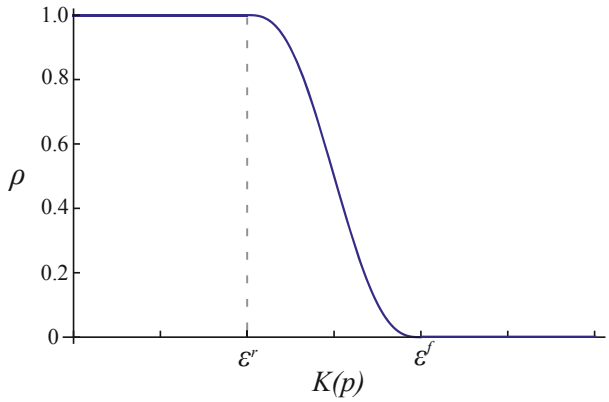


Figure 4.2.1: Restraining function ρ as a function of particle's kinetic energy $K(p)$.

corresponds to a single particle, and is an identity matrix multiplied by the particle's *adaptive inverse inertia* $\phi_i(q_i, p_i)$. We choose $\phi_i(q_i, p_i) = m_i^{-1}(1 - \rho_i(q_i, p_i))$, $1 \leq i \leq N$, where m_i is the particle's mass, and $\rho_i(q_i, p_i) \in [0, 1]$ is a twice-differentiable *restraining function*. When $\rho_i(q_i, p_i) = 0$ (no restraining), $\phi_i(q_i, p_i) = m_i^{-1}$ and the particle is following *full dynamics* (the dynamics derived from the Hamiltonian (4.2.1)). When $\rho_i(q_i, p_i) = 1$ (full restraining), $\phi_i(q_i, p_i) = 0$ and the particle is not moving, whatever the force applied to it. When $\rho_i(q_i, p_i) \in (0, 1)$, the particle smoothly switches between both behaviors.

4.2.4.2 When to switch

We make the restraining function of each particle i depend on its kinetic energy $K_i(p_i) = p_i^2/(2m_i)$, so that $\rho_i(q_i, p_i) = \rho_i(p_i)$. Let ε_i^r and ε_i^f, $\varepsilon_i^r < \varepsilon_i^f$, respectively denote a *restrained-dynamics threshold* and a *full-dynamics threshold* for particle i. We define $\rho_i(p_i)$ as:

$$\rho_i(p_i) = \begin{cases} 1, & \text{if } 0 \leqslant K_i(p_i) \leqslant \varepsilon_i^r, \\ 0, & \text{if } K_i(p_i) \geqslant \varepsilon_i^f, \\ s(K_i(p_i)) \in [0, 1], & \text{elsewhere,} \end{cases} \tag{4.2.15}$$

where $s(\varepsilon)$ is a twice-differentiable function with respect to its argument and, therefore, to p_i, $s(\varepsilon_i^r) = 1$, $s(\varepsilon_i^f) = 0$. One possible choice for this function is an interpolation spline of order five:

$$\eta = \frac{\varepsilon - \varepsilon_i^r}{\delta_i}, \quad s(\varepsilon) = -6\eta^5 + 15\eta^4 - 10\eta^3 + 1, \tag{4.2.16}$$

where $\delta_i = \varepsilon_i^f - \varepsilon_i^r$ is the width of the transition region.

Figure 4.2.1 plots the restraining function ρ as a function of particle's kinetic energy $K(p)$ with the smoothing function $s(\varepsilon)$ defined by eq. (4.2.16).

The inverse mass-matrix is now completely defined. The AR Hamiltonian can then be written as follows:

$$H_{AR}(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T(\mathbf{I}-\rho(\mathbf{p}))\mathbf{M}^{-1}\mathbf{p} + V(\mathbf{q}),$$

where $\mathbf{I}$ is a $3N \times 3N$ identity matrix and $\rho(\mathbf{p})$ is a $3N \times 3N$ diagonal matrix combining the individual $\rho(p_i)$. The equations of motion can be derived from the AR Hamiltonian for each ith particle independently:

$$\begin{aligned} \dot{p}_i &= -\frac{\partial H_{AR}}{\partial q_i} = -\frac{\partial V}{\partial q_i}, \\ \dot{q}_i &= \frac{\partial H_{AR}}{\partial p_i} = \frac{p_i}{m_i}(1-\rho_i(p_i)) - \frac{1}{2}\frac{p_i^2}{m_i}\frac{\partial \rho_i(p_i)}{\partial p_i}. \end{aligned} \tag{4.2.17}$$

As can be seen from the equations, momenta are still propagated as in a full-dynamics simulation (eq. 4.2.2), but positions evolve differently. In particular, when $\rho(q_i,p_i)=1$, $\dot{q}_i = 0$ and the particle is not moving.

In other words, in an adaptively restrained particle simulation, when the module of a particle's momentum becomes small enough (without necessarily becoming zero), the particle's position *completely* stops evolving. Even when a particle is fully restrained, though, its momentum may continue to change, and its kinetic energy might become large enough again for the particle to resume moving. In general, ARPS restrain and release particles repeatedly over time.

Please note that when the inverse inertia matrix is chosen in the described way, the AR Hamiltonian is separable. Therefore, the partitioned Euler integration scheme described above becomes symplectic and explicit.

4.3 Discussion

We now discuss the properties of the AR simulations and illustrate some of them with simple examples.

4.3.1 Phase portrait

Let us discuss in more detail how ARPS modifies the system's trajectories in phase space. For that, we consider a simple case: a single particle in 1D of mass 1 g/mol that is attached with a spring of stiffness 1 kcal/(mol Å^2) to a fixed point (particle with infinite mass). We will then observe the trajectories of this particle in phase space.

We, first, show in Fig. 4.3.1 the phase-space portrait of this system in the full-dynamics case.

Then, in Fig. 4.3.2 we show a phase-space portrait of this system simulated with the AR Hamiltonian ($\varepsilon_1^r = 0.5$ kcal/mol, $\varepsilon_1^f = 0.8$ kcal/mol). Opal lines stand for isolines of this

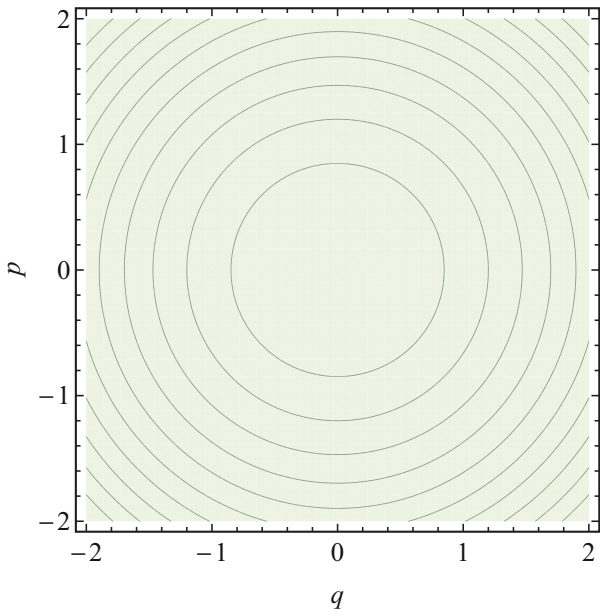


Figure 4.3.1: Phase portrait of the harmonic oscillator in 1D, full-dynamics simulation.

Hamiltonian, the thick black line – for a specific isoline of this Hamiltonian ($H_{AR} \equiv 1$), the dotted red circle represents the corresponding isovalue of the classical Hamiltonian ($H \equiv 1$). The restrained-dynamics region (where the particle is fully-restrained) is blue, the full-dynamics region (where the particle is not restrained) is green, and black dashed lines indicate the boundaries of these two regions. The AR Hamiltonian has modified trajectories: in the restrained-dynamics region positions are constant, but momenta change; in the transition region trajectories smoothly switch between restrained dynamics and full dynamics. In the full-dynamics region trajectories stay unperturbed. When $p = 0$, the AR trajectory is tangential to the classical one.

We performed more AR simulations of this harmonic oscillator, varying the thresholds. The results are shown in Fig. 4.3.3.

As is clear from the plots, the larger the width of the transition region δ, the bigger the distance between dashed lines; the bigger the thresholds ε_1^r and ε_1^f, the larger the blue zone on the plot.

4.3.2 Rescaling time

On the presented above phase portraits of the harmonic oscillator system we do not see time. In reality, the period of the oscillation of the particle will be perturbed by ARPS. We propose a solution to recover this period in this simple case, by considering an AR simulation as a rescaling of time: in the restrained-dynamics part of the trajectory time stops, in the full-dynamics region time is unperturbed, and in the transition region time is adjusted to

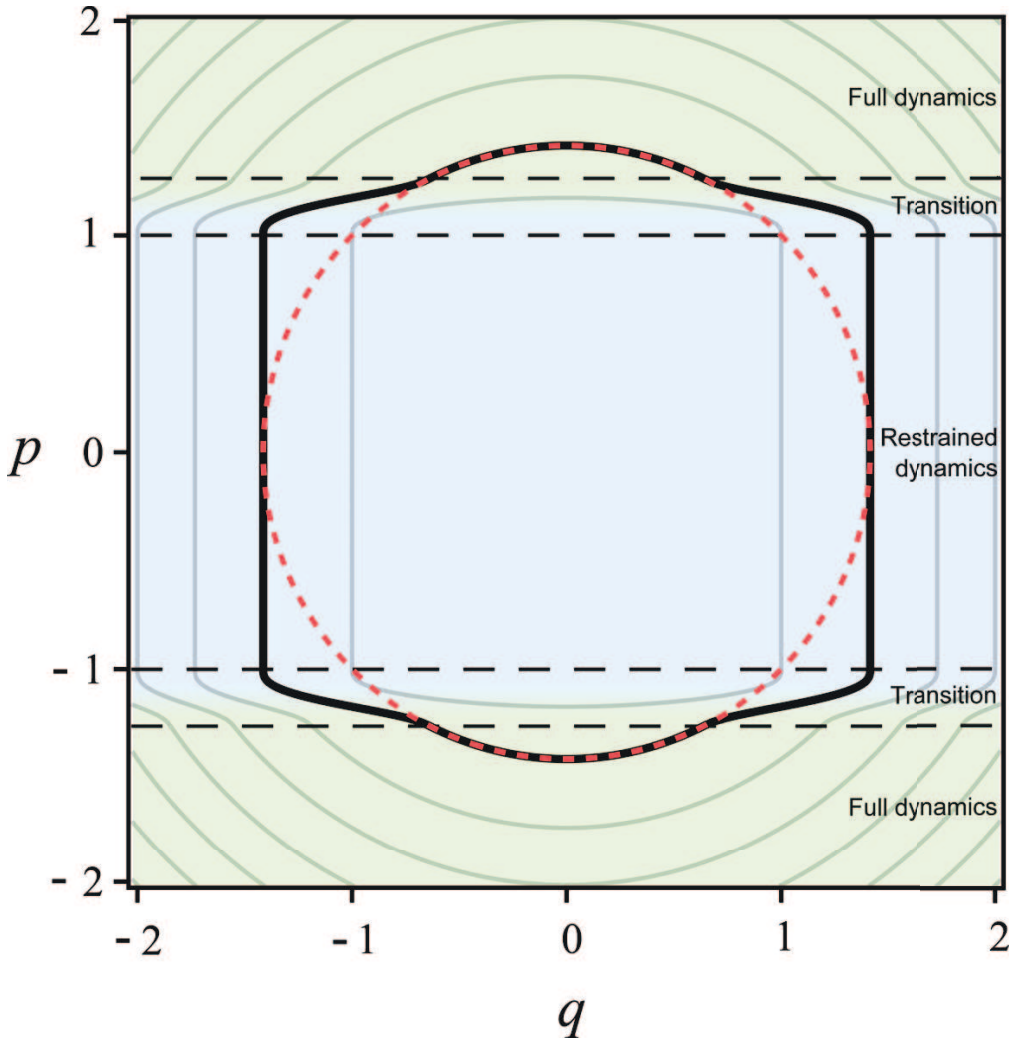


Figure 4.3.2: Phase portrait of the adaptively restrained harmonic oscillator in 1D ($\varepsilon_1^r = 0.5$ kcal/mol, $\varepsilon_1^f = 0.8$ kcal/mol). Opal lines stand for isolines of this Hamiltonian, the thick black line for a specific isoline of this Hamiltonian ($H_{AR} \equiv 1$), the dotted red circle represents the corresponding isovalue of the classical Hamiltonian ($H \equiv 1$). In the full-dynamics region, trajectories remain unperturbed. In the restrained-dynamics region, positions are constant although momenta evolve. In the transition region trajectories smoothly switch between restrained dynamics and full dynamics.

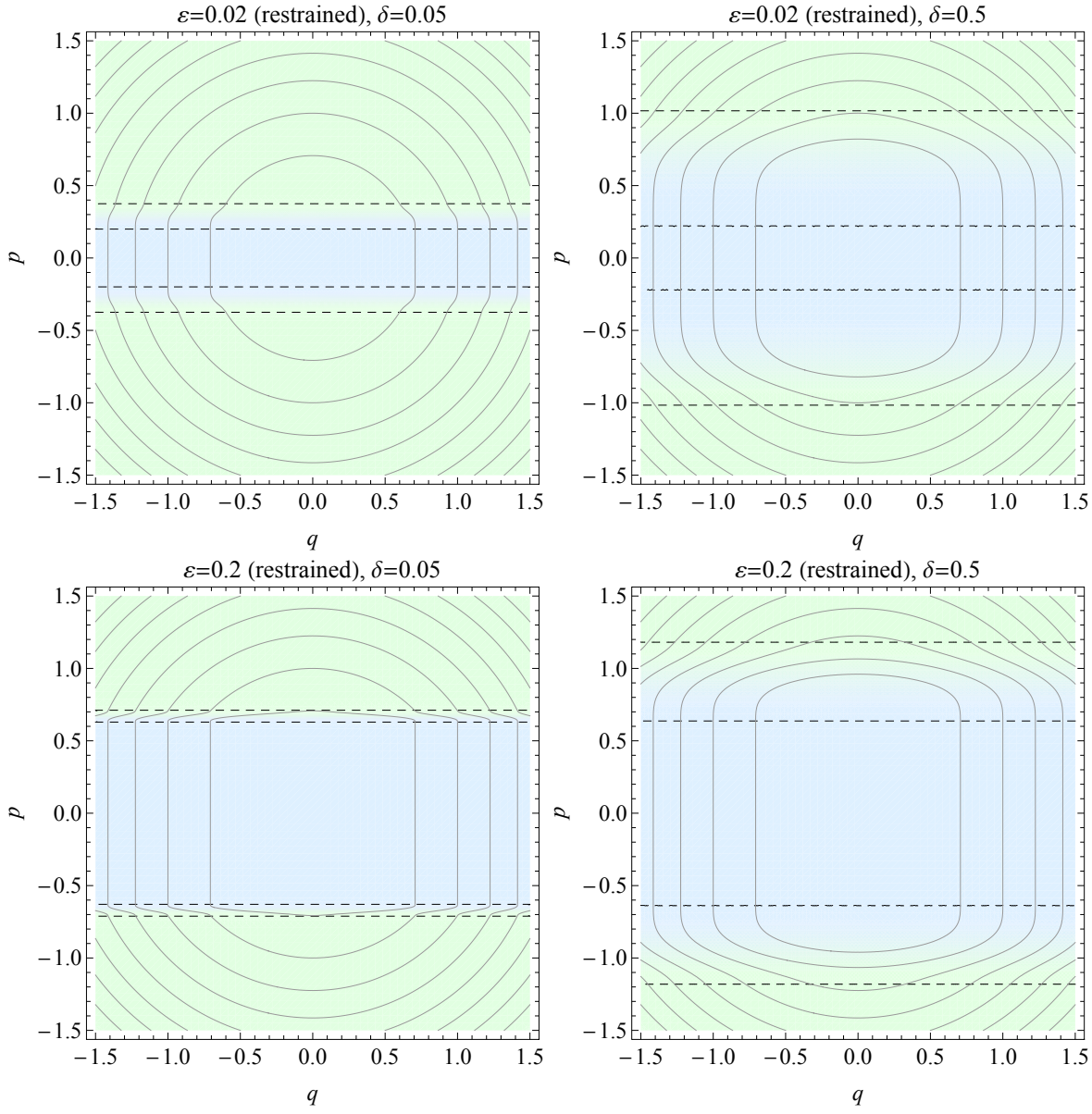


Figure 4.3.3: Phase portraits of the adaptively restrained harmonic oscillator for several sets of thresholds, ε stands for ε_1^r (in kcal/mol), the width of the transition region δ expressed in kcal/mol.

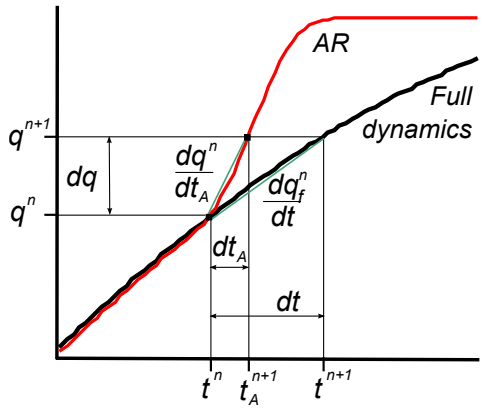


Figure 4.3.4: Projection of the AR trajectory to the full-dynamics trajectory. Detailed description is provided in the text.

interpolate between both behaviors.

To rescale time we do the following. We examine two trajectories: the full-dynamics trajectory – (t^n, q_f^n), and the AR one – (t^n, q^n) (upper indices stand for time steps, the lower index f stands for full dynamics). These trajectories are shown as black and red lines, respectively, in Fig. 4.3.4. We would like to "project" the AR trajectory to the full-dynamics one. For each time step dt we will search for an adaptive time step dt_A, so that, with a step of a Verlet integrator, in the full-dynamics case, the point (t^n, q_f^n) would go to (t^{n+1}, q_f^{n+1}) with the help of momenta p_f^{n+1}, and in AR simulation the point (t^n, q^n) would go to (t_A^{n+1}, q^{n+1}) with the help of momenta p^{n+1}. At time t^n for any n, the positions q^n and the displacements dq^n are the same for both trajectories.

As $\frac{dq^n}{dt} = (1 - \rho(p^n))m^{-1}p^n - \frac{1}{2}p^{nT}\frac{\partial\rho(p^n)}{\partial p^n}p^n$ and $\frac{dq_f^n}{dt_A} = m^{-1}p_f^n$ and $dq^n = dq_f^n$ then, $dt_A(t)$ can be written as:

$$dt_A^n = \frac{(1 - \rho(p^n))m^{-1}p^n - \frac{1}{2}p^{nT}\frac{\partial\rho(p^n)}{\partial p^n}p^n}{m^{-1}p_f^n}dt,$$

From this formula, at each time step we need to calculate p_f^n (the velocity at time t^n on the full-dynamics trajectory). One can do it by integrating the usual Hamiltonian H with an adaptive time step: $p_f^{n+1} = p_f^n + f^n dt_A$, where f^n is the current force. As, therefore, at each integration point the positions q^n are the same for both trajectories, we do not need to recalculate forces for the full-dynamics trajectory at each time step, we just update the full-dynamics momenta and store them.

We have applied this technique to the described harmonic oscillator to recover its oscillation period: performed AR simulations with the rescaled time. The obtained results are

shown in Fig. 4.3.5.

On the top picture of Fig. 4.3.5 we show the situation where the particle is not fully-restrained at the beginning of AR simulation. In this case, we recovered completely the oscillation period and the particle's trajectory with the proposed technique. However, we had to use a very small time step, as the points on the trajectory were no more regular, and when the consecutive points were far from each other we could have lost precision, and the trajectory would have diverged from the real one.

On the bottom picture of Fig. 4.3.5, we show the situation where, at the beginning of the AR simulation, the particle is in the transition region. Thus, we recover only the particle oscillation period, but not its trajectory, as in this case the trajectory does not reach some full-dynamics values q. In this case, we initialize p_f^0 as if we were on a full-dynamics trajectory, with which the kinetic energies are equal: $0.5\left(p_f^0\right)^2/m = 0.5\left(1-\rho\left(p^0\right)\right)\left(p^0\right)^2/m$. Therefore, $p_f^0 = p^0\sqrt{1-\rho\left(p^0\right)}$.

4.3.3 Temperature in the NVT ensemble

Let us now discuss temperature in AR simulations. In this section we consider *only separable Hamiltonians* H_{AR}.

Let the NVT ensemble be sampled with the Langevin thermostat, and let the temperature of the thermostat be set to a certain value T. We would like to determine the corresponding temperature T_{AR} in the AR simulation, which is defined by the formula [214]:

$$T_{AR} = \frac{1}{Dk_B}\left\langle \sum_{i=1}^{N}\left(p_i \cdot \frac{\partial H_{AR}}{\partial p_i}\right)\right\rangle,$$

where D is the dimensionality of the system (*e.g.* $3N$ for N particles in three-dimensional space) and the dot sign stands for the dot product. It is known that:

$$\left\langle \sum_{i=1}^{N}\left(p_i \cdot \frac{\partial H_{AR}}{\partial p_i}\right)\right\rangle = \left\langle \sum_{i=1}^{N}\left(q_i \cdot \frac{\partial H_{AR}}{\partial q_i}\right)\right\rangle = \left\langle \sum_{i=1}^{N}\left(q_i \cdot \frac{\partial V(q)}{\partial q_i}\right)\right\rangle.$$

Thus, the temperature T_{AR} may be represented as a variable that depends only on particle positions. Therefore, it is conserved by AR simulations in NVT ensemble (eq. (4.2.9)) and, as in the full-dynamics simulation, is equal to the thermostat temperature:

$$T_{AR} = T.$$

It is also interesting to study the average of the quantity $K^* = \sum\limits_{i=1}^{N}\frac{p_i^2}{2m_i}$ (the kinetic energy in a full-dynamics simulation) during an AR simulation. For that we introduce an average T^*:

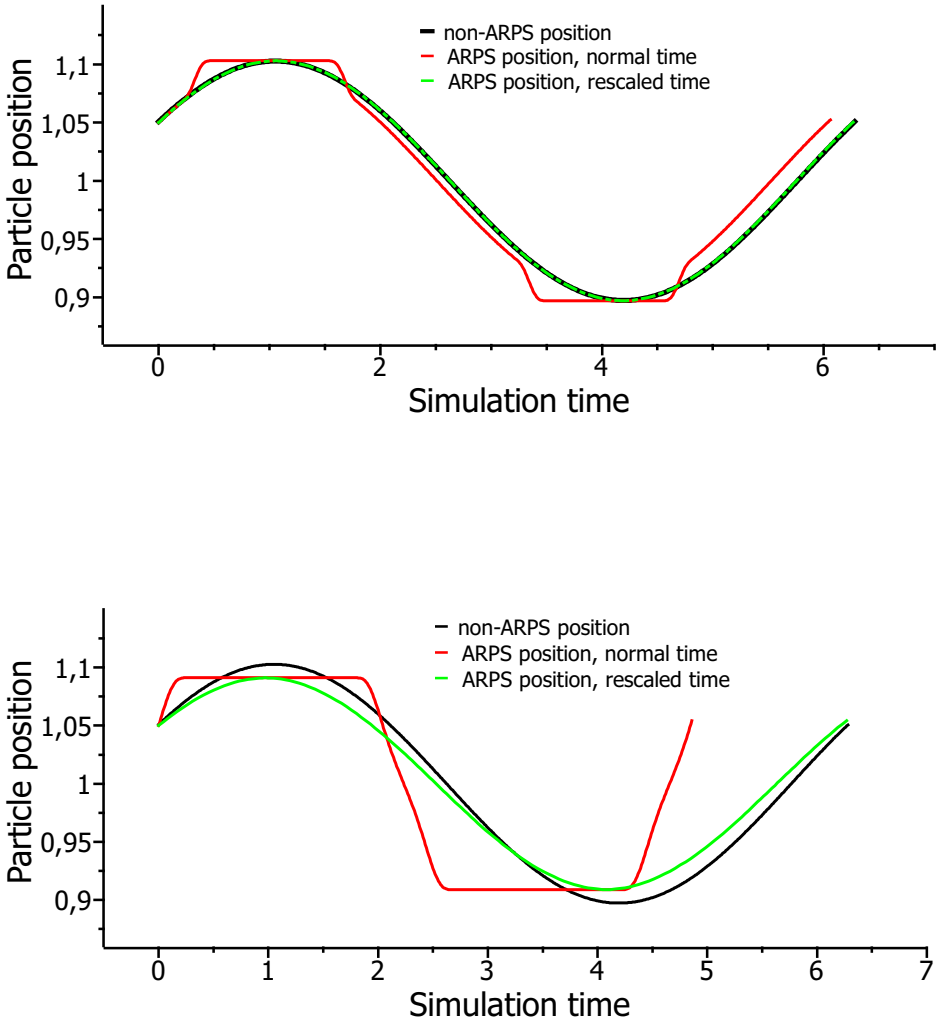


Figure 4.3.5: Period recovering for 1D harmonic oscillator. Top picture parameters: $\varepsilon^f = 0.002$ kcal/mol, $\delta = 0.0015$ kcal/mol. Bottom picture parameters: $\varepsilon^f = 0.005$ kcal/mol, $\delta = 0.0025$ kcal/mol. Initial momentum of the particle was 0.089.

Figure 4.3.6: Dependence of the temperature T^* (eq. 4.3.1) (in AR simulation $\varepsilon^r = 1$ kcal/mol, $\varepsilon^f = 2$ kcal/mol) on the temperature of the thermostat T for a system of 1 particle ($m = 1$) in 3D.

$$T^* = \frac{2}{Dk_B} \frac{\int K^* e^{-\beta H_{AR}}}{\int e^{-\beta H_{AR}}} = \frac{2}{Dk_B} \frac{\int K^* e^{-\beta K_{AR}}}{\int e^{-\beta K_{AR}}}. \tag{4.3.1}$$

To demonstrate the dependence of T^* on the thermostat parameter T for fixed values of both threshold parameters, we considered the following example: a single particle of mass 1 moving freely in 3D. We fixed the values of the thresholds to $\varepsilon^r = 1$ kcal/mol and $\varepsilon^f = 2$ kcal/mol and performed numerical integration of the equation (4.3.1) with Mathematica. The result is displayed in Fig. 4.3.6. It can be seen from the picture, that T^* in this case exceeds the temperature of the thermostat. It happens because even when the particle stops, its momentum continues to evolve, and the quantity K^* depends purely on the particle's momentum.

For the same example, we fixed the thermostat temperature T to 95 K, and varied the values of the two threshold parameters (ε^f was always set to exceed ε^r). The obtained results for T^* are shown in Fig. 4.3.7.

4.4 Algorithms

After having presented and discussed theoretical properties of ARPS, we proceed to a numerical realization of AR simulations.

In this section, we describe an algorithm to integrate in time a set of equations (4.2.17) to simulate a system of particles in the NVE ensemble.

The general scheme is presented in Algorithm 4.1. The function *updateState()* is called every time step. In the Algorithm 4.1 we also provide the pseudo-code for the functions called

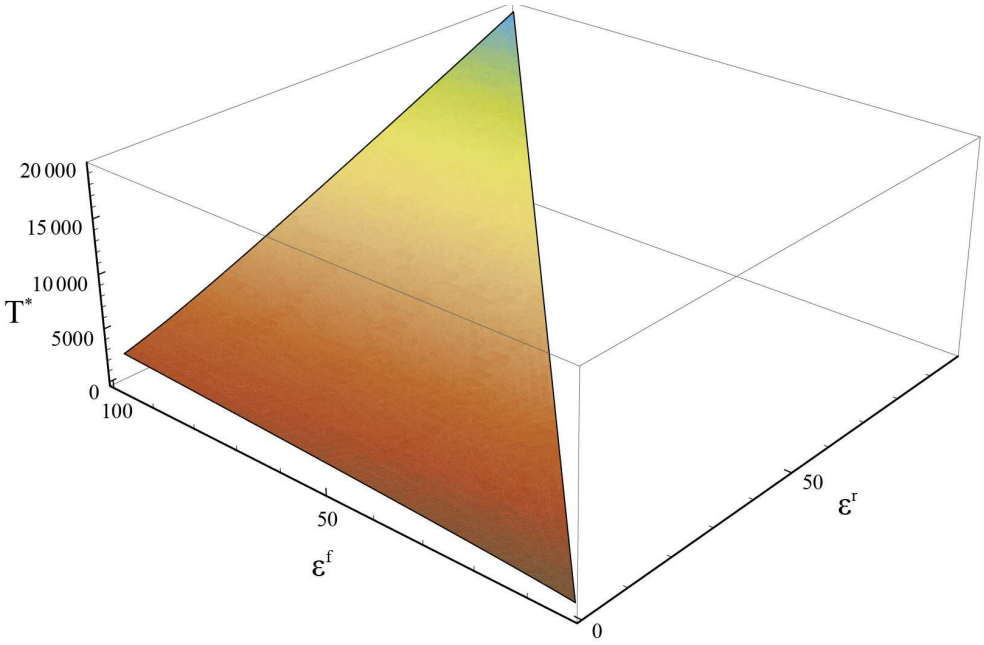


Figure 4.3.7: System of 1 particle ($m = 1$) in 3D. Dependence of the temperature T^* (eq. 4.3.1) on the thresholds ε^r and ε^f for a fixed thermostat temperature $T = 95$ K.

by the *updateState()* function, except for the one that updates the forces. This function will be discussed in more detail later.

The functions *updateMomenta()* and *updatePositions()* represent the symplectic Euler integration scheme (eq. (4.2.12)) applied to the equations (4.2.17).

The function *updatePositions()* can be optimized as follows: q^i_{n+1} (and, therefore, related quantities as, *e.g.* $\partial\rho^i(p^i_{n+1})/\partial p^i_{n+1}$) should be recomputed only for those particles for which $\rho^i(p^i_{n+1})$ is not equal to 1. In other words, for rigid nodes the position $q^i_{n+1} = q^i_n$ and does not need to be updated.

Our approach may accelerate particle simulations due to the smaller number of forces that have to be updated at each time step. This is possible if the interaction potential is written as a sum of terms that depend only on a few relative positional degrees of freedom. This is a frequently-used assumption, an example of such potential is any pairwise potential, *e.g.* Lennard-Jones potential. In this case, the distance, and thus the forces between two fully-restrained particles do not change and, as a result, these forces should not be recomputed. Another common assumption that we consider further on is the fact that the potential is rapidly decaying, in other words, a cutoff distance is introduced.

Further on, we describe two efficient force-update algorithms that benefit from the fact that only some particles in the system are active, and may significantly speed up the calculations. The potential energy can be calculated at the same time as the forces.

Algorithm 4.1 ARPS, NVE ensemble, integration scheme. Lower indices stand for time step numbers, the time step size is denoted by h.

updateState()
  if (time step number $\leqslant$ total number of steps)
    updateForces()
    updateMomenta()
    updatePositions()

updateMomenta()
  for (each particle a^i)
    $p^i_{n+1} = p^i_n + f^i_{n+1} h$

updatePositions()
  for (each particle a^i)
    compute $\rho^i(p^i_{n+1})$ according to equation (4.2.15)
    $$q^i_{n+1} = q^i_n + \left(\frac{p^i_{n+1}}{m^i}(1 - \rho^i(p^i_{n+1})) - \frac{1}{2}\frac{\|p^i_{n+1}\|^2}{m^i}\frac{\partial \rho^i(p^i_{n+1})}{\partial p^i_{n+1}} \right) h$$

4.4.1 Force update, first method

The first force-update method that we propose is based on the examination of the neighbor lists constructed for the system at the current and at the previous time steps.

At each time step we construct a neighbor list for the system: a list of pair of particles being closer than the cutoff distance and considered to be interacting (see Chapter 2). This neighbor list can be constructed in a number of ways. In our implementation, we used an approach described in Ref. [76]. This approach is based on a tree representation of the system and a corresponding hierarchy of axis-aligned bounding volumes [173] (AABB's). Recursively, the left and right sub-trees of internal nodes are searched for interaction depending on the overlap of the corresponding bounding volumes. This hierarchy of bounding volumes, therefore, should be updated every time step.

At the first time step we evaluate all forces for all pairs of particles in the current neighbor list. For any following time step, we examine two lists: the previous neighbor list (constructed at the previous time step) and the current neighbor list. If an interaction is present in the previous list only, then this interaction has disappeared, and we subtract the old force (stored with the previous interaction pair) from the forces of both corresponding particles. If an interaction is present only in the current list, this interaction has just appeared, and we calculate the new force, save it and add it to both particles. If both lists contain an interaction, we check if the radius-vector between the two particles has changed, and only if so, we recalculate the force (subtract the old force and add the new one). This algorithm is optimal in the number of updated forces and should be used if the force calculation is the crucial time for the simulation.

We now give a detailed description of this method.

At each time step we use three lists: current neighbor list, previous neighbor list and a

transition list. An element of any list is a structure (a^i, a^j, r^{ij}, f^{ij}) containing pointers to two particles a^i and a^j, a radius-vector between these two particles r^{ij} and a corresponding vector-force f^{ij} (let it be the force produced by particle a^j on particle a^i). For each particle in the system we store a previous position (position at the previous time step). The total force acting on any particle is the sum of the forces applied to it by all particles that are closer than the cutoff distance to it. For example, for particle a^i the force is:

$$f^i = \sum_{j, r^{ij} < cutoff} f^{ij}.$$

These are the main steps of the algorithm.

1. We compute the current neighbor list. However, while constructing this list we memorize only the distance between the interacting particles and do not compute the corresponding force. As a result, the element of the current neighbor list looks like (a^i, a^j, r^{ij}, 0).

2. We traverse the previous neighbor list. For each list element (a^i, a^j, r^{ij}, f^{ij}), we compare the radius-vector r^{ij} stored in this element to the current radius-vector between the particles a^i and a^j (the difference between their current positions). If these radii are equal (all elements of the vectors are equal, not just their norms), we put this list element to the transition list. Otherwise (the interaction is broken or has changed), we "subtract" from both particles forces corresponding to this interaction. Precisely, from the force of the particle a^i we subtract the force f^{ij} stored in the list element, and we add f^{ij} to the force of the particle a^j. As a result, the forces of particles a^i and a^j are modified as if these particles were no more interacting.

3. We traverse the current neighbor list. For each list element (a^i, a^j, r^{ij}, 0), we compare radius-vector r^{ij} stored in this element to the old radius-vector between the particles a^i and a^j (the difference between the previous positions of these particles). If these radii are not equal (interaction is new or has changed), we calculate the new force f^{ij} corresponding to the interaction between the two particles and add it to the force of the particle a^i and subtract it from the force acting on the particle a^j. We, then, put this list element with the computed force (a^i, a^j, r^{ij}, f^{ij}) to the transition list. As a result, in the transition list we have the elements from the current neighbor list where the interaction has changed compared to the previous time step, and the elements from the previous neighbor list where the interaction has not changed.

4. We swap previous and transition lists. As a result, at the next time step we will consider the transition list produced at the current step as the "previous list". Then current and transition lists are cleared.

Let us now discuss the complexity of this method. Construction of the neighbor list at each time step is linear [76] in time in the number of particles in the system and has a $O(N \log N)$

Algorithm 4.2 ARPS, the first method for incremental force update. Lower indices stand for time step numbers, upper indices are particle indices, a denotes a particle, q denotes its position, f its force, r^{ij} is a radius-vector connecting particles a^i and a^j, and the function *calculateForce()* is specific for each potential.

updateForces_1()

construct a neighbor list, called Current

compare Current and Previous neighbor lists, update forces:

for (each element e_k=(a^i, a^j, r^{ij}_{n-1}, f^{ij}_{n-1}) of the Previous neighbor list)

if ($r^{ij}_{n-1} = q^j_n - q^i_n$)

put e_k to the Transition list

else

// subtract interaction

$f^i = f^i - f^{ij}_{n-1}$

$f^j = f^i + f^{ij}_{n-1}$

for (each element e_k=(a^i, a^j, r^{ij}_n, 0) of the Current neighbor list)

if (r^{ij}_n != $q^j_{n-1} - q^i_{n-1}$)

$f^* = calculateForce(r^{ij}_n)$

put (a^i, a^j, r^{ij}_n, f^*) to the Transition list

// add interaction

$f^i = f^i + f^*$

$f^j = f^j - f^*$

Previous neighbor list = Transition list

clear Current neighbor list and Transition list

(tree) memory complexity. But one is free to choose any other suitable algorithm for this part of calculations. Force update based on the examination of the neighbor lists is linear in time and memory in the neighbor list size (each time step we can reuse the same memory for the lists). This method is optimal in the number of updated forces, but might be too expensive when the cost of updating forces is low.

This method is summarized in pseudo-code in Algorithm 4.2.

If we have periodic boundary conditions for the system (PBC [23]), the algorithm may be changed in its part where the neighbor list is constructed: we may use the algorithm described in Ref. [76] as if the images of the cell were symmetrical replicas of the initial system. Therefore, we have only one current interaction list at each time step. During the examination of the lists, when two radii are being checked for equality, the one that does not come from the neighbor list being traversed, should be recalculated using one-image convention. We need to do this because this distance can be smaller than the cutoff with one of the images of the particles but not with the particle itself. Thus, for PBC we require that the box size is at least twice bigger than the cutoff distance.

4.4.2 Force update, second method

The second method to perform the force update is based on a 3D grid.

To initialize this method, we first create the grid: find the smallest axis-aligned parallelepiped enclosing all system particles, and divide it into cubic cells with an edge size equal to or larger than the cutoff. Then, each particle of the system is mapped to some grid cell according to its position. Each cell has a list of pointers to the particles, each particle stores an index of the cell it belongs to. For each particle in the system we store its previous position (position at the previous time step).

At the first time step we calculate forces for all particles in the system using the grid as follows. Every particle a^i is checked for the proximity with all particles from the current cell and all surrounding cells (26 or less, depending on how close is the current cell to the grid border). This is the sufficient set to find all particles interacting with a specific one. If particle a^i is closer than the cutoff to some particle a^j and $j > i$ we compute the forces acting on both particles due to this interaction[1].

At any other time step for the whole set of active particles (those that $\rho^i < 1$) we perform the three following steps[2], also illustrated in Fig. 4.4.1:

1. **Force update**: For each active particle a^i, we find all particles from the current cell and all surrounding cells, that at the previous time step *were* closer than the cutoff to a^i (using positions from the previous time step for all particles) and that have indices superior to i or are fully-restrained ($\rho^j = 1$)[3]. Then, we calculate the interaction force f^{ij} for each pair that we found and *subtract* this interaction from the particles in the pair (we subtract the force f^{ij} from the force of particle a^i, and we add f^{ij} to the force of particle a^j). We did not lose any previous interaction, as the grid has not yet been updated.

2. **Grid update**: Each active particle a^i is assigned to another grid cell if necessary, according to its current position. At this step the particles do not actually move (their positions are updated in *updatePositions()* function): indices of the containing cells are updated for particles and lists of pointers to particles are updated for grid cells.

3. **Force update**: For each active particle a^i, we find all particles from the current cell and all surrounding cells, closer to it than the cutoff distance and that have indices superior to i or are fully-restrained ($\rho^j = 1$). Then, we calculate the interaction force f^{ij} for each pair that we found and *add* this interaction to the particles in the pair (we add the force f^{ij} to the force of the particle a^i, and we subtract f^{ij} from the force of the particle a^j).

[1] In the general case, any cell size can be chosen. In any case, all particles that interact with a specific particle in cell c can be found in a limited number of cells around cell c.

[2] Only when the first step is performed for all active particles, the second step may be started, etc.

[3] This way, we update interactions between any two active particles a^i and a^j ($i < j$) only once, while considering particle a^i; and we update all interactions between active and fully-restrained particles while considering active particles.

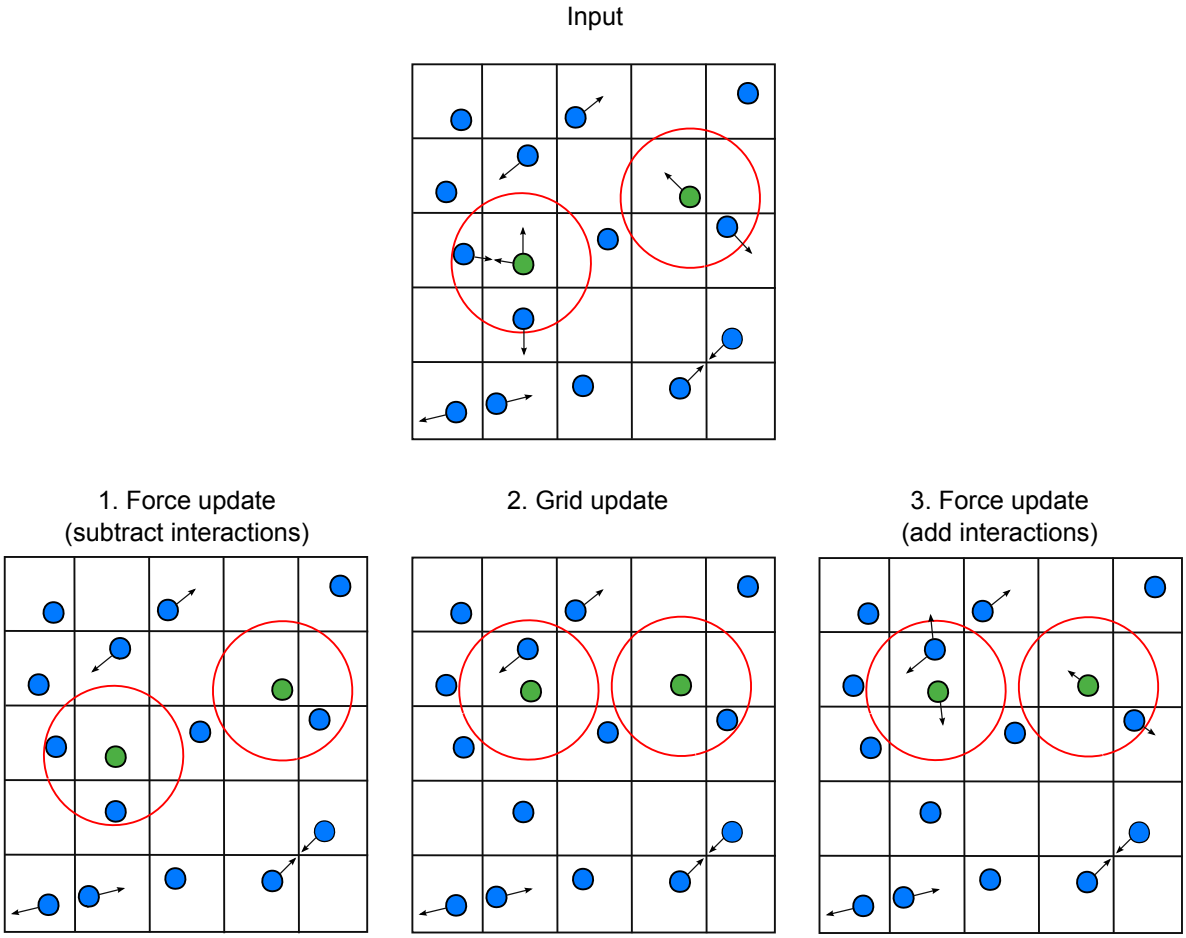


Figure 4.4.1: ARPS. Force update, second method. **Top figure**: an input for the force update algorithm is the system configuration from the previous time step. The interactions between the particles (interaction forces between pairs of atoms) at the previous time step are displayed by arrows. The particles though, are colored according to the restraining function calculated at the *current* time step: active particles are green (light grey), fully-restrained particles are blue (dark grey). **Bottom figure**: the three main steps of the algorithm. **1.** The first step of the algorithm subtracts interactions from the two particles in each pair where at least one particle is active (the corresponding arrows disappear). **2.** The second step of the algorithm updates the grid according to the positions of active particles at the current time step (only these particles have moved). **3.** The third step of the algorithm adds interactions due to the new positions of the active particles to both particles in each pair where at least one particle is active (new arrows corresponding to the new forces appear).

Algorithm 4.3 ARPS, the second method for the incremental force update. In this Algorithm a denotes a particle, q denotes its position, f its force, r_n^{ij} is a radius-vector connecting particles a^i and a^j at time step n, and the function *calculateForce()* is specific for each potential.

updateForces_2()

if (step =1)
  create grid, distribute all particles to the grid cells
  for (each particle a^i)
    for (each particle a^j from the current cell and the surrounding cells)
      if ($r^{ij} \leqslant$ cutoff **and** $i < j$)
        //add interaction
        $f^* = calculateForce(r^{ij})$
        $f^i = f^i + f^*$
        $f^j = f^j - f^*$

else
  for (each *active* particle a^i)
    for (each particle a^j from the current cell and the surrounding cells)
      if ($r_{n-1}^{ij} \leqslant$ cutoff **and** ($\rho_n^j = 1$ **or** $i < j$))
        //subtract previous interaction, previous time step positions are used
        $f^* = calculateForce(r_n^{ij})$
        $f^i = f^i - f^*$
        $f^j = f^j + f^*$
  for (each *active* particle a^i)
    if necessary, assign the particle to another grid cell
  for (each *active* particle a^i)
    for (each particle a^j from the current cell and the surrounding cells)
      if ($r_n^{ij} \leqslant$ cutoff **and** ($\rho_n^j = 1$ **or** $i < j$))
        //add new interaction, current positions are used
        $f^* = calculateForce(r_n^{ij})$
        $f^i = f^i + f^*$
        $f^j = f^j - f^*$

In this method we calculate more forces than in the first one (up to two times more), but the time complexity is linear in the number of active particles in the system, which was more adapted to the numerical examples that we provide. The memory complexity is the size of the grid, and is thus linear in the number of particles in the system. The summary of this method can be found in pseudo-code in Algorithm 4.3.

If periodic boundary conditions are required, the algorithm may be slightly modified. One possibility (that we use in our implementation) is to create in the grid a surrounding layer of empty cells (buffer cells), and pre-store an index of another grid cell which is a "mirror cell" from the PBC point of view to this one (along the axis or axes where we crossed the border of the simulation box, we subtract/add a box size). Then, for each cell we still examine all surrounding cells, and if one of these surrounding cells is a buffer cell, when searching for

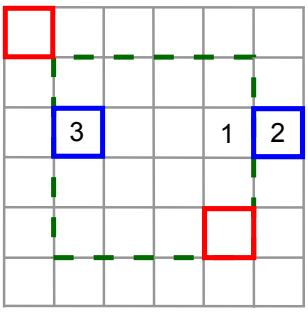


Figure 4.4.2: Possible modification of a 2D-grid in the second force-update method to introduce periodic boundary conditions. The green dashed line surrounds the grid cells that may be occupied by the system's particles, and the cells outside this line (buffer cells) are always empty. Highlighted cells of the same color (red and blue) represent mutual mirror cells. If during the algorithm's execution, while examining neighboring cells of some cell (*e.g.* cell 1 on the figure), a buffer cell is found (cell 2), its mirror cell (cell 3) will be considered instead.

interactions, we will address particles from its "mirror cell". A graphic explanation for this phenomena in 2D is shown in Fig. 4.4.2.

4.5 Results

In this section, we provide several numerical experiments that illustrate ARPS and demonstrate its advantages.

The ARPS method was implemented in C++ and tested either on Computer 1 (two Intel Xeon X5450 3GHz quad-core processors, with 16 GB of RAM, on a Windows Vista 64-bit operating system) or on Computer 2 (one Intel 2.40 GHz quad-core processor with 4GB of RAM, on a Windows Vista 32-bit operating system). The implementation was serial, so that each simulation only used one processing core at a time. For the force update, the second method proposed in Section 4.4.2 was used. As a reference, full-dynamics simulation we considered an AR simulation with both thresholds equal to 0.

4.5.1 Argon liquid

To show that ARPS is able to produce long stable trajectories, we simulated a periodic 3D box of 21 952 Argon particles (box length 99.3048 Å, particle mass 39.95 g/mol) in the NVE ensemble. A Lennard-Jones potential [215] was used:

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

where r is an inter-particle distance. The following parameters were used [215]: $\epsilon/k_B = 120$

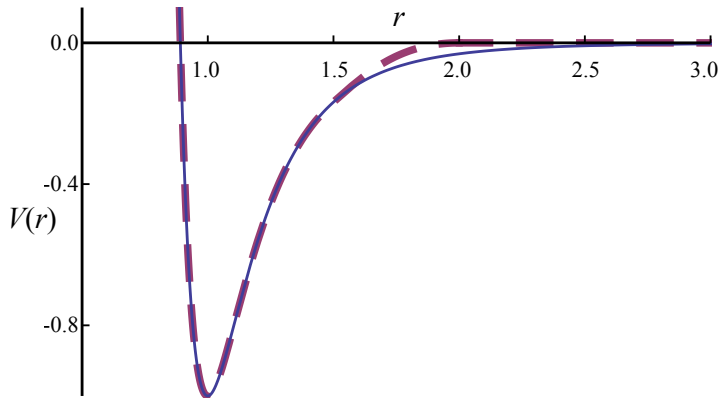


Figure 4.5.1: Lennard-Jones potential, normal $V_{LJ}(r)$ (firm line) and truncated between 1.5 and 2 Å $V_{LJ}^{tr}(r)$ (dashed line).

K, $k_B = 0.00198721$ kcal/mol/K, $\sigma = 3.4$Å, cutoff $c = 8$Å, and the potential was truncated through a smoothing function applied between 7.5 and 8 Å (switching distance d_s is, thus, in this case equal to 0.5 Å). Precisely, the potential that we used was $V_{LJ}^{tr}(r) = V_{LJ}(r)w(r)$, where $w(r)$ is a switching function:

$$c_1 = \left(1/\left(c^2 - d_s^2\right)\right)^3$$
$$c_2(r) = c^2 - r^2, \quad c_3 = 3\left(c^2 - d_s^2\right)$$
$$c_4(r) = c_2(r)\left(c_3 - 2c_2(r)\right)$$

$$w(r) = \begin{cases} 1, & r^2 < d_s^2; \\ c_2(r)c_4(r)c_1, & d_s^2 \le r^2 \le c^2; \\ 0, & r^2 \ge c^2. \end{cases}$$

Normal $V_{LJ}(r)$ and truncated $V_{LJ}^{tr}(r)$ Lennard-Jones potentials are shown in Fig. 4.5.1.

After a small period of equilibration, we launched four simulations: a reference (full-dynamics) simulation, and three AR simulations with different thresholds (time step size 0.488 fs, 5 000 000 time steps, total simulation time 2.44 nanoseconds, Computer 1). Figure 4.5.2 shows, first, the system's *total adaptive energy* for both AR simulations, *i.e.* the value of H_{AR} over time. These adaptive energies are stable, demonstrating that ARPS can produce long trajectories. We would like to stress that we did not use any thermostat to prevent energy drift. Adaptively restrained simulations in the NVE ensemble are stable because they derive from a Hamiltonian, for which symplectic integrators are readily available (*e.g.* partitioned Euler method that we use in the current implementation). Moreover, for the inverse inertia matrix that we have chosen, the AR Hamiltonian is separable and, therefore, the partitioned Euler method becomes explicit.

Note that even though the initial conditions are identical in all simulations, the adaptive Hamiltonians are different from the reference Hamiltonian, and are also different from each other, because the modified inverse inertia matrices are not the same. As a result, the conserved quantities (the values of the various Hamiltonians), *i.e.* the total energy in the reference case, or the total adaptive energies in the AR cases, are different from each other.

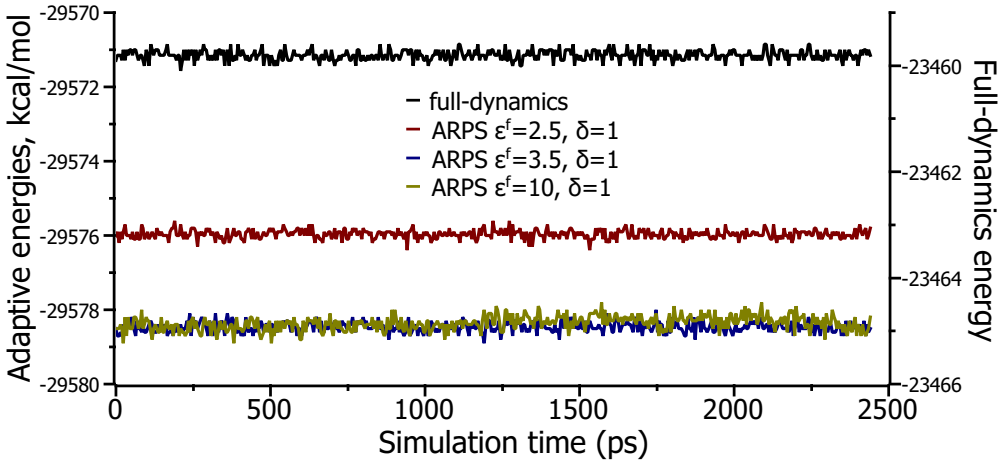


Figure 4.5.2: Four simulations of a periodic 3D box of 21 952 Argon particles (NVE ensemble): reference simulation and 3 AR simulations with different pairs of thresholds (in kcal/mol). We output total full-dynamics energy (in kcal/mol) for the reference simulation and total adaptive energies for AR simulations.

In Fig. 4.5.3, for all four simulations we plot the number of the forces updated each time step (top) and the number of active particles at each time step (down). They are stable during all the simulations.

We were also interested in the evolution of the system's linear momenta during the simulation. To study that, in Fig. 4.5.4 we output the momenta along each axis ($p_x = \sum_{i=1}^{N} p_x^i$, p_y and p_z) and the total momenta ($p_{tot} = p_x + p_y + p_z$) for the full-dynamics simulation (top) and one AR simulation (down). One can see that AR simulation did not perturb this linear momenta a lot compared to the full-dynamics one.

The information about the simulations is summarized in Table 4.1. For each AR simulation, we report the thresholds that were used, the average number of updated forces $\langle N_F \rangle$ and its percentage relatively to the reference simulation, the average number of active particles $\langle N_{act} \rangle$ and this number as a percentage of the total number; the average computation time $\langle t \rangle$ per time step in ms, and the total speedup (due to the reduction in $\langle N_F \rangle$) with respect to the reference simulation (which took about 10 CPU days).

4.5.2 Collision cascade

To demonstrate how AR particle simulations allow us to smoothly trade between computational cost and precision, we simulated the collision cascade in 1D, 2D and 3D.

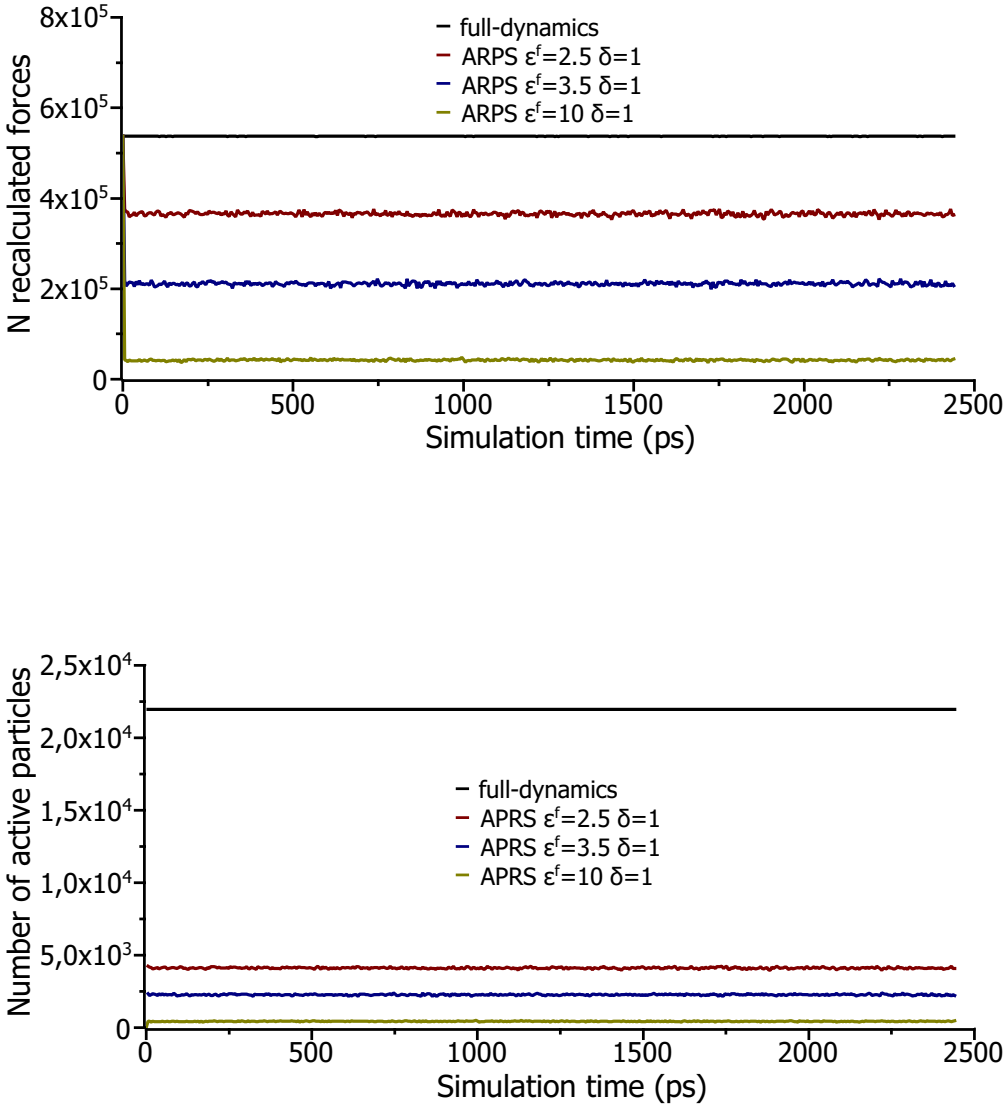


Figure 4.5.3: Four simulations of a periodic 3D box of 21 952 Argon particles, with different pairs of thresholds (in kcal/mol) for AR simulations (NVE ensemble): number of forces updated each time step (top) and number of active particles at each time step (down).

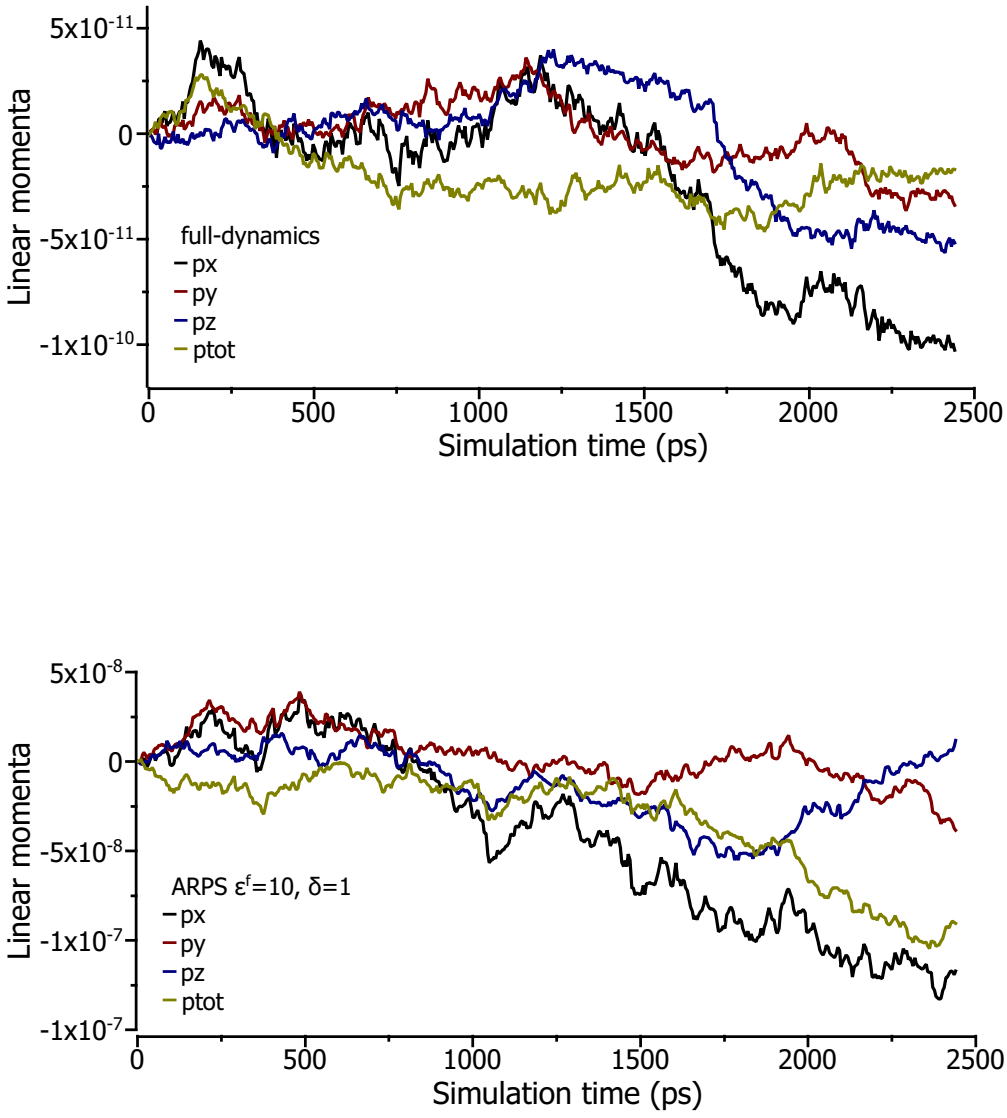


Figure 4.5.4: Simulations of a periodic 3D box of 21 952 Argon particles (NVE ensemble): linear momenta evolution for the full-dynamics simulation (top) and one AR simulation (down).

Simulation	Threshold, kcal/mol	$\langle N_F \rangle$	Speedup, N_F	$\langle N_{act} \rangle$	$\langle t \rangle$, ms	Speedup, time
Reference	——	537 186	——	21 952	180	——
ARPS 1	$\varepsilon^f = 2.5$, $\varepsilon^r = 1.5$	365 942 (68%)	$\approx$ 1.5x	4116.2 (19%)	89	$\approx$ 2x
ARPS 2	$\varepsilon^f = 3.5$, $\varepsilon^r = 2.5$	211 682 (39%)	$\approx$ 2.5x	2272.3 (10%)	57	$\approx$ 3x
ARPS 3	$\varepsilon^f = 10$, $\varepsilon^r = 9$	43 203 (8%)	$\approx$ 12.5x	434.5(2%)	15	$\approx$ 12x

Table 4.1: Summary of the information on four simulations of a periodic 3D box of 21 952 Argon particles, with different pairs of thresholds for AR simulations (NVE ensemble). Average number of forces updated each time step $\langle N_F \rangle$, average number of active particles $\langle N_{act} \rangle$, average wall-clock time per time steps $\langle t \rangle$ may be significantly reduced.

4.5.2.1 Collision cascade in 1D

First, we simulated a 1D collision cascade in the system of 100 particles of equal mass (1 g/mol) in the NVE ensemble. Precisely, into a freely evolving system, at some time step, we injected kinetic energy: increased the momenta of the most left particle. We, then, measured the shock front at each time step – the sequence number (from the left) of the particle with the largest velocity (time derivative of the position) among all particles in the system.

This test was performed on Computer 2, with a time step of 0.048 fs, until the shock reached the most right particle. Particles were considered interacting through a Lennard-Jones potential ($\sigma = 1$, $4\epsilon = 0.001$, cutoff = 5.25 Å). We performed a full-dynamics simulation and two AR simulations with different sets of thresholds. Initial temperature before shock in the full-dynamics simulation was 3.73 K. The shock was applied at time step 1000. We considered two cases: a relatively large shock (Fig. 4.5.5, top) so that the temperature became 620 K on average in the full-dynamics simulation and a relatively small shock (Fig. 4.5.5, down) so that the temperature became 8 K on average in the full-dynamics simulation.

As can be seen in Fig. 4.5.5, the shock front in AR simulations was propagated very accurately, especially in the one with the large shock. When a small shock is considered, at the end of simulation, we can see slight differences between the reference simulation and the AR simulations.

The information about these three simulations is summarized in Tables 4.2 and 4.3, for the large and the small shock respectively. As can be seen, the simulation time can be significantly reduced with ARPS, as this is the case where ARPS is beneficial: a small part of the system is more active than all the rest at each time step.

However, we can obtain even bigger acceleration of the simulation in 2D.

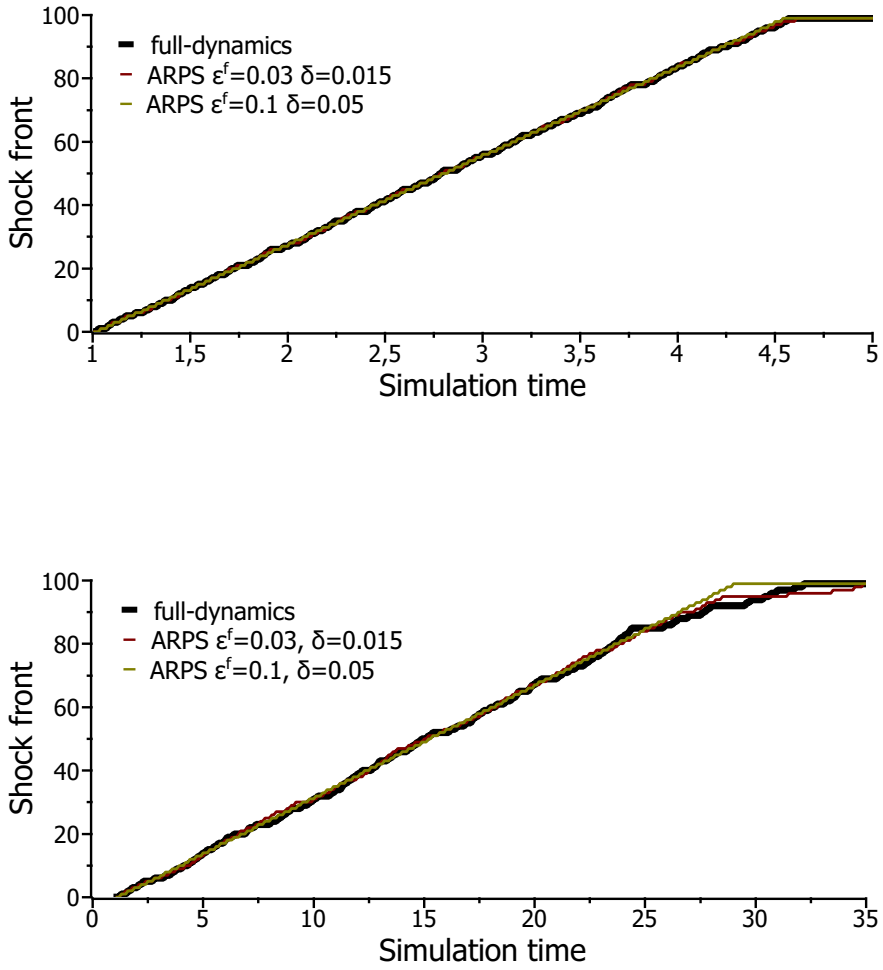


Figure 4.5.5: Simulating a collision cascade in the 1D system of 100 particles in the NVE ensemble (thresholds in kcal/mol). Shock front for a large shock (top) and a small shock (down).

Simulation	Thresholds, kcal/mol	$\langle N_{act} \rangle$	$\langle N_F \rangle$	Total wall-clock time, s
Reference	—	100	435	0.36
ARPS 1	$\varepsilon = 0.03$, $\delta = 0.015$	21.5	324	0.27 (1.3x)
ARPS 2	$\varepsilon = 0.1$, $\delta = 0.05$	1.37	24	0.15 (2.4x)

Table 4.2: Summary of the information on three simulations of the collision cascade in 1D, NVE ensemble. Large shock.

Simulation	Thresholds, kcal/mol	$\langle N_{act} \rangle$	$\langle N_F \rangle$	Total wall-clock time, s
Reference	—	100	421	1.92
ARPS 1	$\varepsilon = 0.03$, $\delta = 0.015$	21	272	1.35 (1.4x)
ARPS 2	$\varepsilon = 0.1$, $\delta = 0.05$	1.25	21	0.28 (6.9x)

Table 4.3: Summary of the information on three simulations of the collision cascade in 1D, NVE ensemble. Small shock.

4.5.2.2 Collision cascade in 2D

We then simulated a collision cascade in a 2D system composed of $N = 5\,930$ particles with mass 1 g/mol in the NVE ensemble, using again the Lennard-Jones potential ($\epsilon/k_B = 120$ K, $\sigma = 3.4$Å, cutoff = 8 Å, the potential was truncated through a smoothing function applied between 7.5 and 8 Å). We performed four simulations of the shock induced by a particle launched at high velocity towards the initially static 2D system: a reference (full-dynamics) simulation, and three AR simulations with varying degrees of precision (time step size 0.0488 fs, 7 000 time steps, total simulation time 342 fs, Computer 2). Figure 4.5.6 compares the final configurations reached by the four simulations. For each AR simulation, we output the maximal absolute displacement of the particles $\Delta q_{\max} = \max_i \|q_i - q_i^f\|$ (where q_i is a vector of coordinates of particle i at the last time step of the AR simulation and q_i^f is the vector of coordinates of this particle at the last time step of the reference simulation) and the root-mean-square deviation (RMSD):

$$RMSD = \sqrt{\frac{\sum_{i=1}^{N} \|q_i - q_i^f\|^2}{N}}.$$

In this example, AR simulations allow for large speedups (up to 10x) while preserving the features of the shock extremely well.

The video showing the four complete simulations in wall-clock time can be found on the NANO-D group website or by following the link. This video demonstrates how AR simulations accelerate the collision cascade in the described system, and, in particular, one can see that AR simulations are very efficient at the beginning of the test, when only few particles are

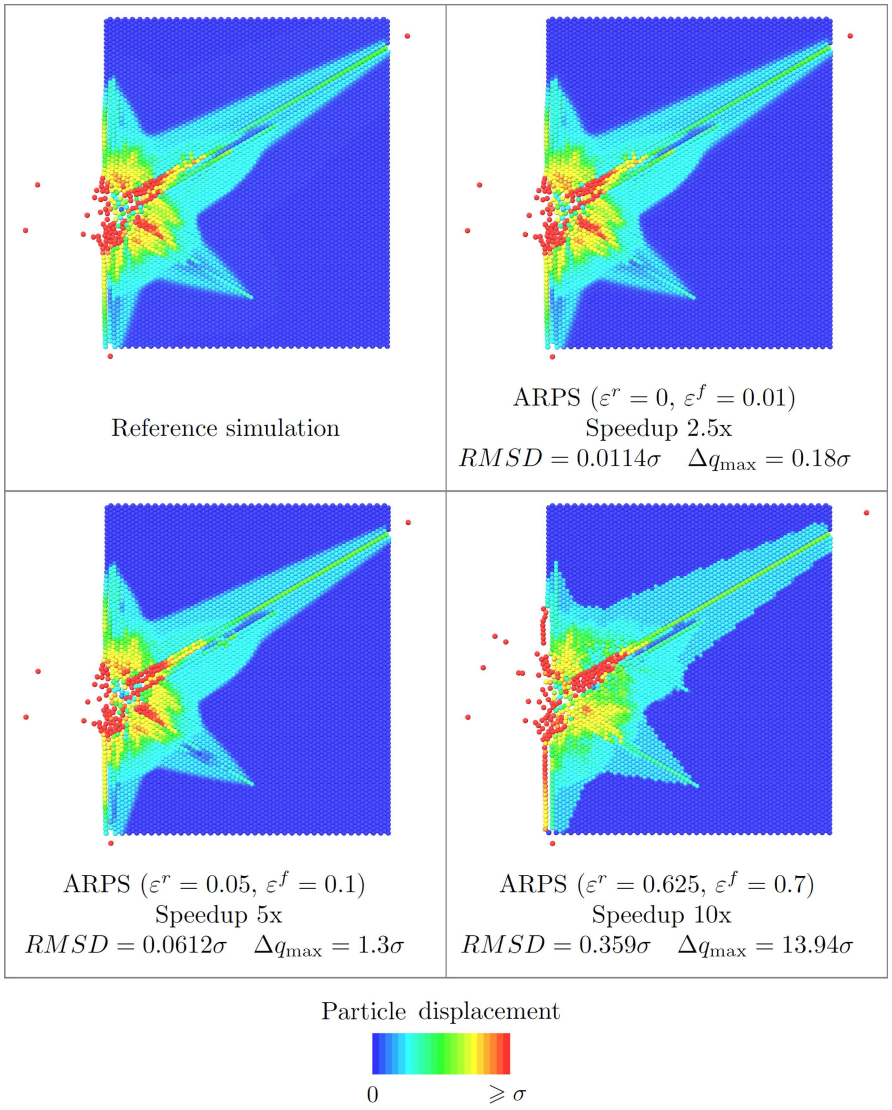


Figure 4.5.6: Simulating a collision cascade in 2D in the NVE ensemble with controlled precision (thresholds in kcal/mol). Adaptively restrained simulations allow us to smoothly trade between precision and speed. Even for large speedups (up to 10x) the features of the shock are extremely well preserved.

reached by the shock, and all simulations have quite a similar speed when almost all the particles in the system are activated by the shock.

Error analysis. We will now discuss the error provided by ARPS due to the simplification. Already in Fig. 4.5.6 we output RMSD and $\Delta q_{\max}$ for the AR simulations. However, it might be interesting to analyze the distribution of the error in space. For that, in the middle picture of Fig. 4.5.7, for each AR simulation, we output in color the absolute displacement A_i of the particles from their final positions in the reference simulation:

$$A_i = \begin{cases} \dfrac{\|q_i - q_i^f\|}{\sigma}, & \text{if } \|q_i - q_i^f\| < \sigma; \\ \sigma, & \text{else.} \end{cases}$$

For visual clarity at the top picture of Fig. 4.5.7 we display particle displacements in the same format. It can be seen that the error appears as in the zones were the particles move the most, as on the border of the shock wave.

In the bottom figure of Fig. 4.5.7 we output in color the relative error, error in percents R_i, of the particle displacement in AR simulations relative to its displacement in the reference simulation:

$$R_i = \begin{cases} \dfrac{\|q - q_i^f\|}{\|q_i^f - q_i^0\|} \times 100\%, & \text{if } \dfrac{\|q - q_i^f\|}{\|q_i^f - q_i^0\|} < 0.01; \\ 1\%, & \text{else,} \end{cases}$$

where q_i^0 is the initial position of the particle i. As can be seen, few particles have the relative error exceeding 1% even for the fastest simulation.

In this example, the 2D plane was static before the shock. Note however that, even if the plane particles had been initially moving, we could have chosen threshold values that would have allowed us to freeze a significant number of these particles, and speed up the simulation. What is important, in order to obtain a large speedup while preserving well the shock features, is that in the system of interest a small part of the system is moving relatively rapidly compared to the rest of the system.

We now increase even more the dimensionality of the simulation and consider a 3D system.

4.5.2.3 Collision cascade in 3D

In this test, we simulated a 3D system of 138 916 particles of mass 1 g/mol, interacting through the same Lennard-Jones potential as in the previous example, in the NVE ensemble. This system was pre-minimized as a parallelepiped of dimensions: $201.842 \times 103.6 \times 234.374$ Å. One particle was then set apart and, given some initial momenta, launched in the direction of the initially static parallelepiped. We performed four simulations for 48.88 fs with the time step 0.000488 fs on Computer 2: one reference simulation and three AR simulations. We present the results in Fig. 4.5.8 (top), drawing only a part (approximately a half) of particles in the 3D parallelepiped for visual clarity. We again output the RMSD and $\Delta q_{\max}$

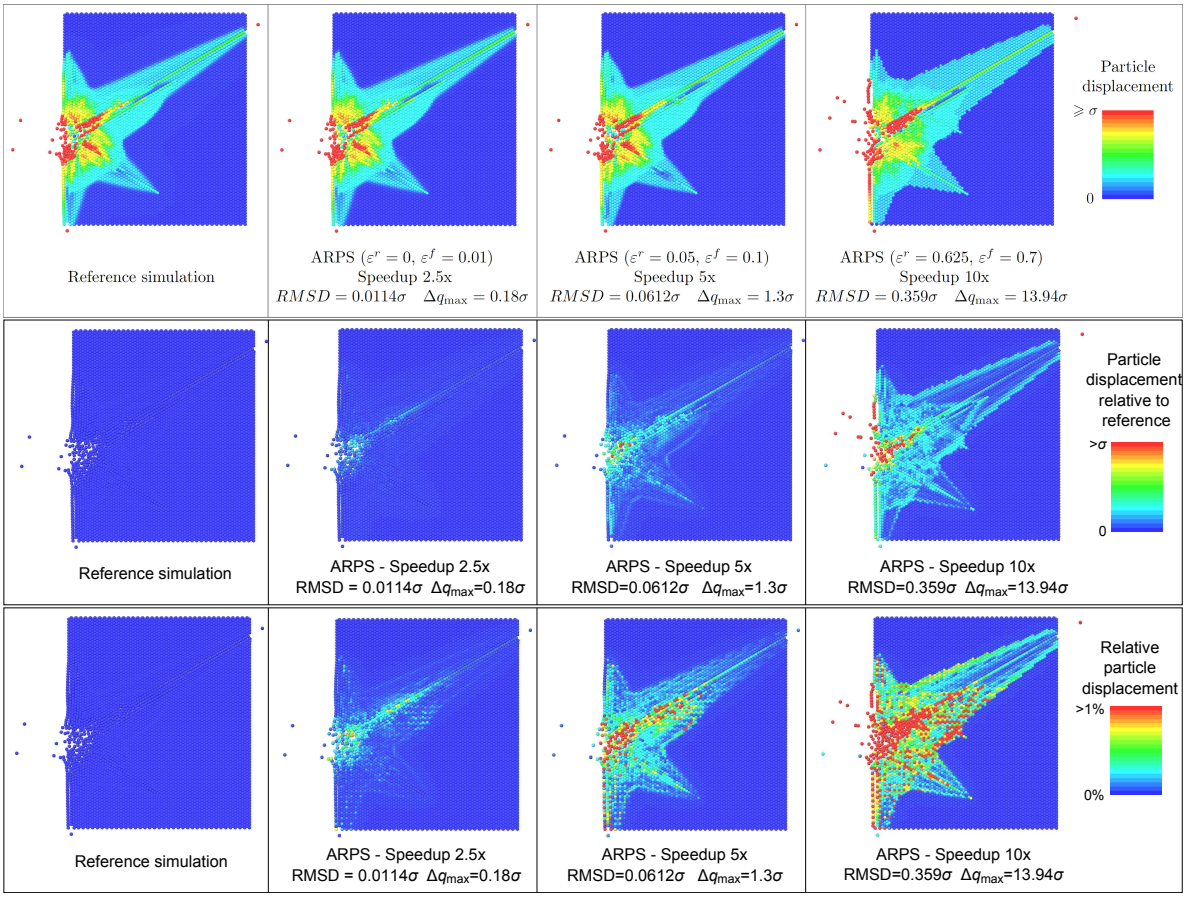


Figure 4.5.7: Simulating a collision cascade in a 2D system in the NVE ensemble, error analysis: particle displacements A_i (top), particle displacements relative to reference (middle); particle displacements R_i relative to the displacements in the reference simulation(bottom). Thresholds are expressed in kcal/mol. Error appears as in the zones were the particles move the most, as on the border of the shock wave.

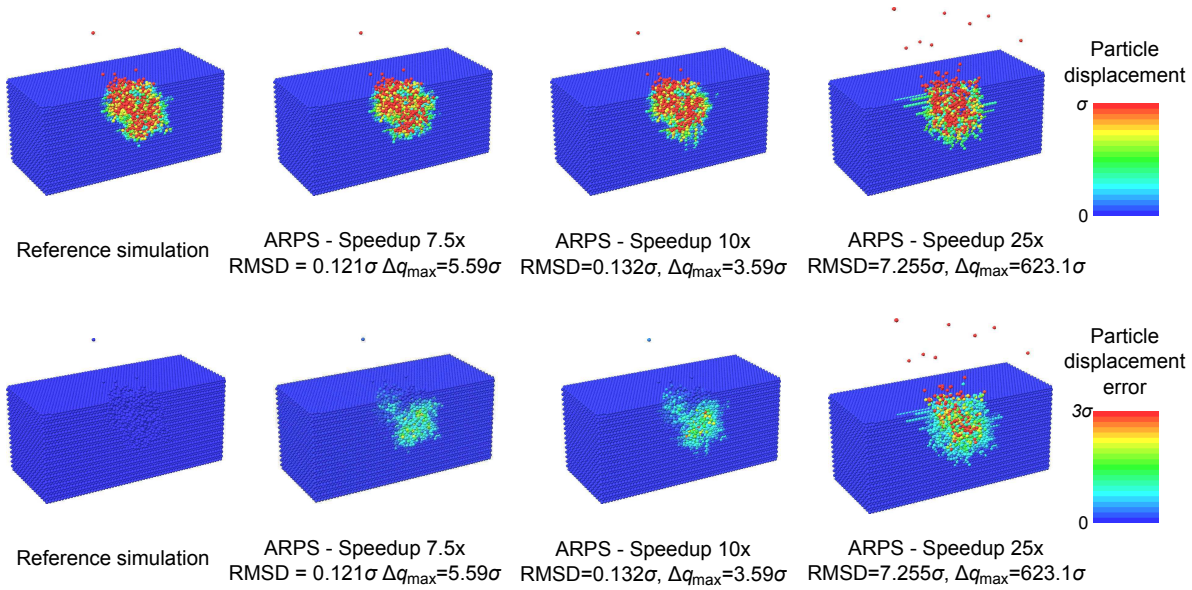


Figure 4.5.8: Collision cascade in a 3D system of 138 916 particles in the NVE ensemble. For visual clarity only a half of the particles is displayed. Adaptively restrained simulations allow us to smoothly trade between precision and speed. Thresholds for AR simulations from left to right (in kcal/mol): $\varepsilon^r = 0$, $\varepsilon^f = 0.01$ (speedup 7.5); $\varepsilon^r = 0.01$, $\varepsilon^f = 0.02$ (speedup 10); $\varepsilon^r = 99$, $\varepsilon^f = 100$ (speedup 25).

for each simulation to estimate the produced error. Figure 4.5.8 (bottom) shows, as in the previous example, the absolute displacement A_i of the particles from their final positions in the reference simulation.

As can be seen from the figure, big speedups can be achieved but, as before, for larger accelerations of the simulation, the error increases as well.

To summarize, even though with ARPS the dynamics of the system is modified and system's static equilibrium properties can not be easily corrected with unbiasing techniques in NVE ensemble, ARPS allow users to smoothly trade between the simulation's speed and precision and to obtain approximate results faster, *e.g.* the main features of the collision cascade are preserved with ARPS while simulation time is significantly reduced.

We will now discuss several tests performed in the canonical (NVT) ensemble.

4.5.3 Radial distribution function

We first show that ARPS may be used to obtain correct statistical properties in the NVT ensemble. We consider a statistics dependent only on positions, so we do not have to unbias it: the Radial Distribution Function $g(r)$ (RDF). In statistical mechanics, a radial distribution function in a system of particles describes how the particle density varies as a function of

the distance from a reference particle. This function plays an important role in molecular simulations because macroscopic thermodynamic quantities can be calculated using RDF.

The radial distribution function, *i.e.* the probability to find some particle in the shell dr at the distance r of a reference particle, may be written as follows:

$$g(r) = \frac{dn(r)}{4\pi r^2 \rho \, dr}, \tag{4.5.1}$$

where $dn(r)$ is the number of particles at a distance between r and $r+dr$ from the reference particle, and $\rho = N/V$ is the density of the system.

In practice, we were computing distances r_{ij} between all particles closer than 20 Å and putting values $1/\|r_{ij}\|^2$ to a histogram. This histogram was normalized by the value $4\pi\rho d N/2$, where d is the width of the histogram bin (the multiplier $4\pi\rho d$ comes directly from the formula (4.5.1), and $N/2$ comes from the fact that, in this case, we had N reference particles and counted each distance between the particles twice).

We performed Langevin simulations of a system of 343 Argon particles interacting in a 3D periodic box through a Lennard-Jones potential used in the Argon liquid example but with the cutoff 8.2 Å, and the potential was truncated through a smoothing function applied between 7.7 and 8.2 Å (box length 25.56 Å, Langevin friction coefficient $\gamma = 1$, $T = 94.4$ K). The particles were initially far from equilibrium, positioned at the nodes of a 3D cubical lattice (to make the comparison meaningful: if the system is pre-equilibrated, the RDF obtained with an AR simulation is good anyway since we stay for long periods in correct conformations and average them). We launched a full-dynamics simulation and an AR simulation ($\varepsilon^f = 40$ kcal/mol, $\delta = 0.5$ kcal/mol) and computed the RDF $g(r)$ in each case (time step size 0.488 fs, 150 000 time steps, total simulation length 73.2 ps, Computer 2). For both simulations, we started collecting statistics when the number of updated forces in the AR simulation became non-zero (t=4.88 ps). Otherwise we accumulate too much non-significant initial configurations for AR simulations. Figure 4.5.9 plots the results: even though the AR simulation has significantly modified the system's dynamics (only 3% of the particles were moving on average at each time step), the curves coincide, demonstrating that the equilibrium statistics have been preserved.

The AR simulation modifies the system's dynamics significantly. As a result, convergence to the correct radial distribution function $g(r)$ is slower in the AR simulation than in the full-dynamics simulation *for a given number of time steps*. To illustrate this, we plot in Fig. 4.5.10 the reference and the AR radial distribution functions obtained after 50 000 steps. As can be seen, RDF obtained with the AR simulation is noisier than the reference one for this number of time steps – precisely, because particle positions haven't been sufficiently sampled, the plotted AR RDF keeps traces of the original cubical lattice used as initial configuration. Thanks to the reduced number of moving particles, though, the computational cost of an AR time step is significantly reduced: AR simulation is about 8 times faster than the reference one.

The goal was, however, to show that the AR RDF converges exactly to the reference RDF.

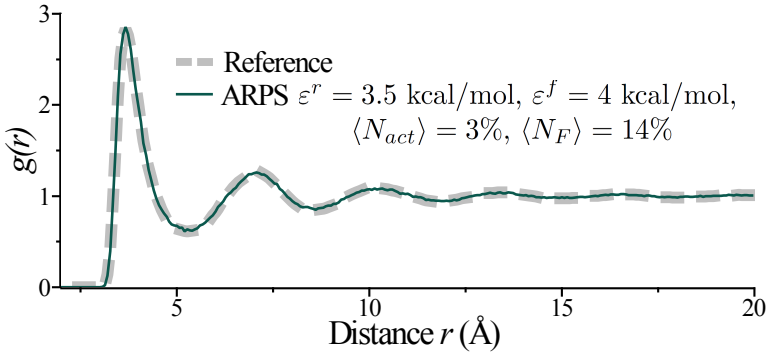


Figure 4.5.9: Radial distribution functions for an Argon system (periodic 3D box, 343 particles, NVT ensemble). Even though the AR simulation has significantly modified the system's dynamics (see $\langle N_{act} \rangle$), the curves coincide.

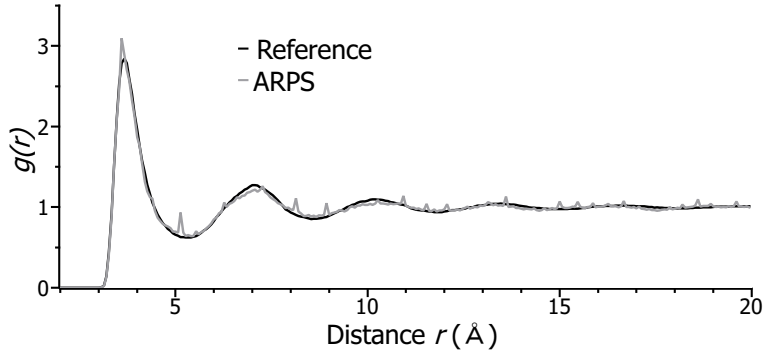


Figure 4.5.10: Radial distribution functions of an Argon system obtained with full-dynamics and AR simulations (periodic 3D box, 343 particles, NVT ensemble). After 50 000 simulation steps RDF in the AR simulation is noisier than the reference one.

Note, that for an AR simulation in the NVE ensemble, the correct RDF would in general not be obtained, since the system is being simplified and some distances between the particles in the system may become inaccessible.

4.5.4 Polymer in solvent

Finally, to show how ARPS may be used to obtain static properties faster than with full-dynamics simulations, we performed Langevin simulations of a toy polymer in the small solvent box, in order to predict the polymer's hydrodynamic radius:

$$\frac{1}{R_H} = \frac{1}{N^2}\left\langle \sum_{i\neq j} \frac{1}{r_{ij}} \right\rangle_H ,$$

where N is the number of monomers in a polymer, r_{ij} is the distance between the i-th and j-th monomers, and $\langle \dots \rangle_H$ is the ensemble average (*i.e.* the average of the values obtained at each step of one simulation). Please note that $\langle \dots \rangle_H = \langle \dots \rangle_{H_{AR}}$ in this case as the hydrodynamic radius depends only on particles positions:

$$\frac{1}{R_H} = \frac{1}{N^2}\left\langle \sum_{i\neq j} \frac{1}{r_{ij}} \right\rangle_H = \frac{1}{N^2}\left\langle \sum_{i\neq j} \frac{1}{r_{ij}} \right\rangle_{H_{AR}} .$$

This numerical experiment is representative of numerous applications of particle simulations in physics, chemistry, biology, etc., where information is collected about a small part of the particle system, but where the rest of the system is required so that the collected information is realistic (*e.g.* simulating an active site in an enzyme, a solute passing through a membrane channel, a defect in a graphene sheet, a crack in a material, etc.).

We modeled the solvent as 343 Argon particles in a 3D box of length 25.56 Å interacting through Lennard-Jones potential with the same set of parameters as for the Argon liquid example, and represented the polymer as 5 particles of mass 40 g/mol each, connected by springs (stiffness $k = 30\,000\epsilon/\sigma^2$, equilibrium length $d_0 = 0.7071 \times \sigma = 2.4$Å). The solvent interacted with the polymer through the same Lennard-Jones potential. Since we wanted to obtain statistics on the polymer, *we only restrained the solvent particles* ($\varepsilon^r = 18$ kcal/mol, $\varepsilon^f = 20$ kcal/mol).

We then compared the rate of convergence of the polymer's hydrodynamic radius R_H to its average value $\langle R_H \rangle$ in full-dynamics simulations and AR simulations. To achieve this despite the intrinsic stochasticity of Langevin simulations, we performed 90 day-long full-dynamics simulation and 90 day-long AR simulations (time step size 0.488 fs, Langevin friction coefficient $\gamma = 1$, $T = 350$ K, a total of 6 CPU months, Computer 1), and compared the variance of R_H in both cases. The obtained $\langle R_H \rangle$ values are very close (4.459, and 4.462 respectively).

Figure 4.5.11 plots the variance of R_H in full-dynamics simulations and AR simulations, first as a function of simulation time (top) and then as a function of wall-clock time (bottom). In simulation time (*i.e.* per time step), the full-dynamics variance decreased faster than

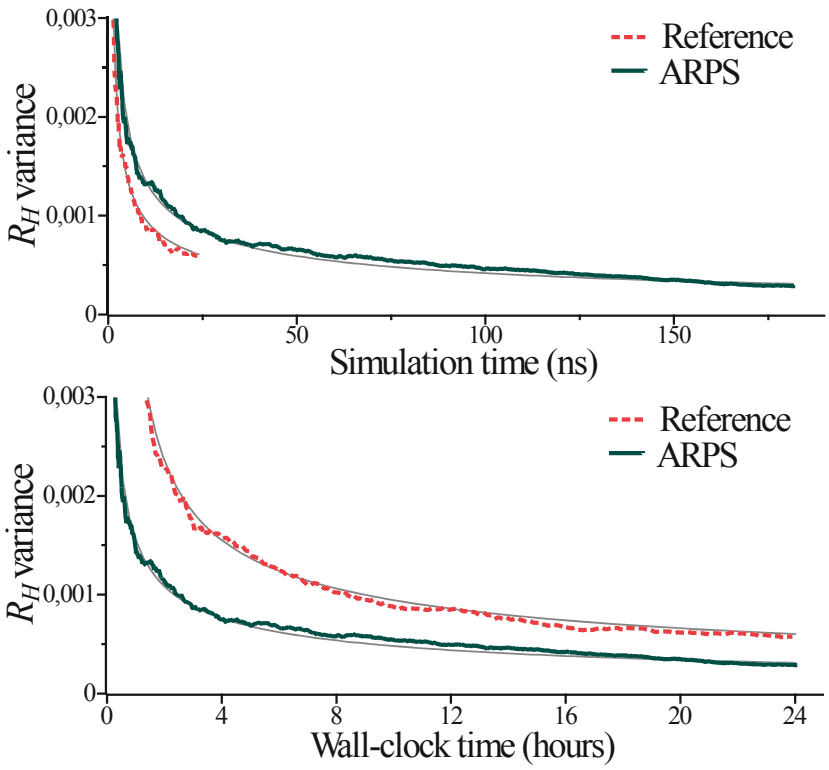


Figure 4.5.11: Computing the hydrodynamic radius R_H of a solvated polymer, NVT ensemble. Full-dynamics simulations reduce the variance more at each time step (top), but AR simulations perform many more time steps, so that they reduce the variance faster in wall-clock time (bottom). For any target precision, AR simulations compute the hydrodynamic radius four times faster than full-dynamics simulations.

the AR variance. Since solvent particles, which influence the polymer conformations, were sometimes immobile in the AR simulations (2% of active particles on average), full-dynamics simulations sampled phase space more efficiently at each time step. The computational cost of AR time steps, however, was significantly reduced, so that AR simulations allowed for about 7x more time steps *in the same wall-clock duration* (3 645 000 versus 495 000). As a result, the AR variance decreased faster than the full-dynamics variance *per wall-clock second.* To estimate the speedup, we fitted the curves with functions of the form: $a/\sqrt{x+b}$ (thin grey lines on the plots). We obtained $a = 0.0029$ ($b = -0.484$) for the reference simulations and $a = 0.0015$ ($b = -0.065$) for the AR simulations, indicating that the speedup permitted by AR simulations to reach a specific precision was around 4x.

The variance of the hydrodynamic radius does not converge exactly to the zero value, because of the error occurring due to the finite time step size.

Simulating polymer in solvent in NVT ensemble, we observed not just the hydrodynamic

Average over 90 simulations	Reference	ARPS	Error	Speedup
$\langle \Delta r_{\max} \rangle$	3.77258	3, 77688	0.1%	4x
$\langle R_G \rangle$	2.77471	2.7776	0.1%	4x
$\langle R_E \rangle$	6.82529	6.83596	0.2%	4x
$\langle R_H \rangle$	4.45913	4.46185	0.06%	4x

Table 4.4: Average values of the polymer parameters over reference and AR simulations, relative errors for these parameters, and the speedups. For each parameter the speedup was computed, as described in the text, from the coefficients a of the fitting functions of the form $a/\sqrt{x+b}$.

radius R_H, but also the average maximal distance of a monomer from the polymer's center of mass $\langle \Delta r_{\max} \rangle_H = \langle \Delta r_{\max} \rangle_{H_{AR}}$, the radius of gyration R_G:

$$R_G = \sqrt{\frac{1}{N} \sum_i \left\langle (r_i - R)^2 \right\rangle_H} = \sqrt{\frac{1}{N} \sum_i \left\langle (r_i - R)^2 \right\rangle_{H_{AR}}},$$

where r_i is the position of the i-th monomer and $R = \sum_i r_i / N$ is the polymer's center of mass, and the end-to-end distance R_E:

$$R_E = \sqrt{\left\langle (r_N - r_1)^2 \right\rangle_H} = \sqrt{\left\langle (r_N - r_1)^2 \right\rangle_{H_{AR}}}.$$

In Table 4.4 we output average values of the variables of interest, where this average is taken over 90 simulations, and also the relative error: the difference between the full-dynamics and AR values divided by the reference value.

Therefore, we have shown that our method is able to obtain unbiased positional static properties, and does it faster than full-dynamics simulations[4].

The video corresponding to this numerical experiment can be found on the NANO-D group website or by following the link. It shows the full-dynamics simulation of the described system and several AR simulations with different sets of thresholds.

Active density Simulating a toy polymer in a solvent, we saw that, at the beginning of the simulation, active particles were mostly concentrated around the moving protein. This happened because the protein was put into the solvent unfolded, and it was tending to fold. To determine whether this phenomena persists in long term, we studied system's *active density*, the percentage of particles that are active around the polymer. In practice, for each solvent particle we measured its distance to the polymer's center of mass, and added to a corresponding histogram bin the degree of restraining (the value $\rho^i \in [0, 1]$) for this particle. The

[4]Implicit solvent methods would probably still be faster, but would probably produce significant changes in, *e.g.* gyration radius [93].

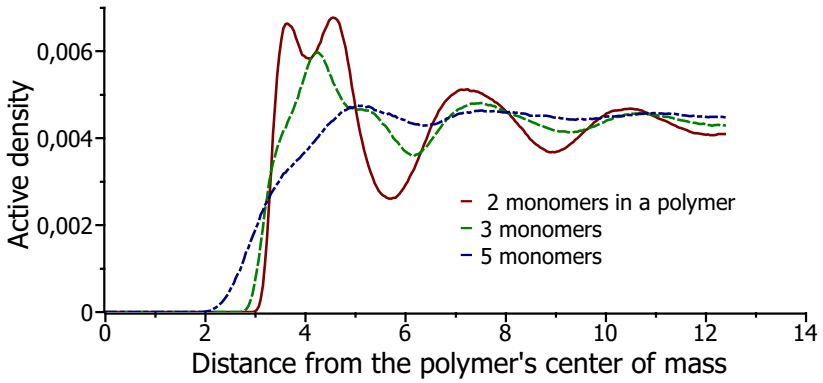


Figure 4.5.12: Active density of the solvent for the cases of 2,3 and 5 monomers in a polymer (NVT ensemble). The solid line stands for 2 monomers, the dashed line for 3 monomers, the dotted line for 5 monomers.

histogram was then normalized. The results for the solvent composed of 343 Argon particles (same parameters for the solvent and the simulation, total simulation steps 13 000), and the polymer with 2, 3 and 5 monomers (same parameters as above), respectively, are shown in Fig. 4.5.12.

We also performed an AR simulation of the same system ($\varepsilon^f = 4$ kcal/mol, $\delta = 0.5$ kcal/mol). We observed that the active densities in the full-dynamics case coincide with the those obtained with ARPS. Therefore, the probability of the solvent particle to be active does not depend on its distance from the polymer center of mass, there are no more active particles around the polymer than somewhere else in the simulation box.

Let us give an explanation for this effect. We can write the active density $N_{act}(r, dr)$ at the spherical volume situated between r and $r + dr$ from the polymer's center of mass as follows:

$$N_{act}(r, dr) = \int I(r, dr)\rho(p)e^{-\beta H_{AR}(q,p)}dp\,dq,$$

where $I(r, dr)$ is the indicator function saying whether the particle is situated in the interval $[r, r + dr)$ of distances from the polymer's center of mass.

As the AR Hamiltonian is separable, we can write:

$$N_{act}(r, dr) = N(r, dr)\langle\rho\rangle,$$

where $N(r, dr)$ is the number of particles in the spherical volume bounded by $[r, r + dr)$, where $\langle\rho\rangle$ is the average value of the restraining function among all the particles in this volume. This $\langle\rho\rangle$ does not depend on r as it was integrated separately, and the histogram

was normalized. This is the reason why the active density is the same for the AR and the reference simulation, even though $\langle\rho\rangle$ might be different for these simulations.

4.6 Conclusion

In conclusion, we proposed a novel general algorithm accelerating particle simulations due to the smaller number of forces to be updated at each time step. It is based on a new, adaptively restrained Hamiltonian that switches positional degrees of freedom on and off during the simulation. We have shown, in the case of AR molecular dynamics, how NVE and NVT simulations may be performed, and how static equilibrium properties can be obtained. We believe our approach may be extended in numerous directions, since the inverse mass matrix $\Phi(\mathbf{q}, \mathbf{p})$ can be chosen according to the specific needs of different types of simulations. Moreover, with any such matrix, ARPS can be coupled to numerous other complementary methods (*e.g.* those including long-range interactions [74, 75]). New incremental algorithms for the force update may have to be designed for them. It can also directly use numerous existing accelerating techniques [97, 85, 98].

In the future we would like to study more possible applications of this approach, *e.g.* studying mixtures, phase transitions, etc. We also want to explore several theoretical aspects of ARPS: the choice of thresholds, their influence on the system's dynamical properties, connections to Monte Carlo methods, etc. Analytical solutions for some parts of the simulation might be obtained and multiple-time-stepping techniques [77] may be applied.

Chapter 5

Hierarchical adaptively restrained particle simulations

Dans le Chapitre précédent, nous avons présenté une nouvelle approche générale qui accélère les simulations moléculaires: les simulations de particules restreintes de façon adaptative. Cette approche est fondée sur un Hamiltonien restreint de façon adaptative à l'aide d'une matrice d'inertie inverse modifiée. Nous avons proposé un choix particulier de cette matrice, et nous avons démontré plusieurs avantages des simulations adaptatives produites sur quelques exemples.

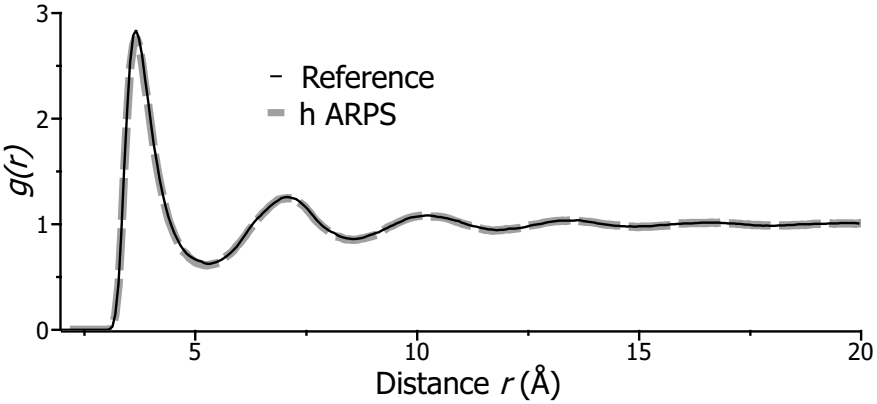
Dans ce Chapitre, nous continuons à travailler sur cette approche, et nous proposons un autre choix de la matrice d'inertie inverse dans le Hamiltonien H_{AR}. Cette matrice active et désactive les degrés de liberté positionnels relatifs dans le système, et se prête bien aux systèmes représentés sous forme d'arbre binaire (l'utilité de cette représentation et des algorithmes se fondant sur elle ont été abordés en détail au Chapitre 3). Nous appelons cette approche "Simulations hiérarchiques de particules restreintes de façon adaptative" (ARPS hiérarchiques).

Nous proposons un algorithme d'intégration efficace pour une simulation de particules dans ce cas, et démontrons cette approche sur plusieurs exemples numériques.

In the previous Chapter, we have presented a novel general approach that accelerates molecular simulations: Adaptively Restrained Particle Simulations. This method relies on an adaptively restrained Hamiltonian, i.e. a Hamiltonian with a modified inverse inertia matrix. We have proposed one particular choice of this matrix and have demonstrated several advantages of the obtained simulations on several examples.

In this Chapter, we continue to work on this approach, but we propose an alternative choice for the inverse inertia matrix in the AR Hamiltonian. This matrix adaptively turns on and off relative positional degrees of freedom in the system (and not the positional degrees of freedom of individual particles), and is designed to simulate the systems represented hierarchically, e.g. as binary trees (the use of such representations, as well as some algorithms relying on them, were discussed in detail in Chapter 3). We call this approach hierarchical Adaptively Restrained Particle Simulations (hierarchical ARPS).

After introducing theoretical aspects of hierarchical ARPS, we propose an integration scheme to perform efficient particle simulations in this case, and provide several numerical examples illustrating this approach.



5.1 Introduction

In the previous Chapter, we have introduced a novel general approach to speed up particle simulations. This approach relies on an adaptively restrained (AR) Hamiltonian, and the equations of motion for a system of particles may be derived from this function. The AR Hamiltonian involves a modified inverse inertia matrix, which allows us to switch positional degrees of freedom on and off repeatedly during the simulation. This inverse inertia matrix can be chosen in a number of ways to suit various applications and particularities of the system of interest.

In the previous Chapter, we have proposed one specific choice of the inverse inertia matrix: this matrix was designed to produce simulations in Cartesian coordinates, where particles stopped and resumed moving repeatedly over time, and the decision about the switch was taken for each particle independently. This inverse inertia matrix was, therefore, diagonal, and easy to store and manipulate.

The restraining function for each particle (the function that determined whether this particle should be moving or not) was chosen to be dependent on the particle's momenta or, more precisely, its kinetic energy. This function had as its parameters two user-defined thresholds to regulate the amount of simplification of the particle's motion. It was possible to use different thresholds for different particles, or for different groups of particles. In particular, we could force a part of the system to stay permanently active and the rest of the system to follow AR dynamics with a user-defined amount of simplification (as in the polymer-in-solvent study described in Section 4.5.4).

In this Chapter, we introduce an alternative choice for the inverse inertia matrix in the AR Hamiltonian. This matrix is designed to simulate systems that may naturally be represented as a hierarchy, and precisely, as a binary tree (the use of such representations and the algorithms relying on them were discussed in detail in Chapter 3). In this new approach that we call hierarchical Adaptively Restrained Particle Simulations (hierarchical ARPS), we switch on and off *relative* positional degrees of freedom in the system, and not just the positional degrees of freedom of individual particles. This results in large blocks of particles being restrained *together*, moving together and being split repeatedly during the simulation. The forces inside rigid blocks do not need to be recomputed at each time step. This inverse inertia matrix is, however, no more diagonal in general case, and might be dense.

The restraining function for each particle in this case depends not only on the properties of this particle, but also on the properties of some other particles. As in classical ARPS, two thresholds are used to tune the simplification of the system's motion, and a subsystem may be forced to stay permanently active during the simulation.

The rest of this Chapter is organized as follows. First, we present the theoretical aspects of the new approach and discuss its properties. Then, we propose an efficient integration algorithm to perform hierarchical AR simulations. Finally, hierarchical ARPS is illustrated on several numerical examples.

5.2 Theory

Let us now describe the theoretical basis of the hierarchical ARPS approach.

As in the previous Chapter, we consider that in the reference full-dynamics simulation, the system of N particles in 3D is associated to a Hamiltonian $H(\mathbf{q}, \mathbf{p})$:

$$H(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T \mathbf{M}^{-1} \mathbf{p} + V(\mathbf{q}). \tag{5.2.1}$$

Correspondingly, the evolution of positions $\mathbf{q}$ and momenta $\mathbf{p}$ of the particles in a reference simulation is described by the following equations:

$$\begin{aligned} \dot{\mathbf{p}} &= -\frac{\partial H}{\partial \mathbf{q}} = -\frac{\partial V}{\partial \mathbf{q}}, \\ \dot{\mathbf{q}} &= \frac{\partial H}{\partial \mathbf{p}} = \mathbf{M}^{-1}\mathbf{p}. \end{aligned} \tag{5.2.2}$$

We, again, consider the adaptively restrained Hamiltonian H_{AR} for the system of particles in its general form:

$$H_{AR}(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T \Phi(\mathbf{q}, \mathbf{p})\mathbf{p} + V(\mathbf{q}), \tag{5.2.3}$$

where the modified inverse inertia, $3N \times 3N$ matrix $\Phi(\mathbf{q}, \mathbf{p})$ adaptively switches system degrees of freedom on and off. The precise form of this matrix should be chosen.

5.2.1 Choice of inverse mass matrix

We now describe the choice of matrix $\Phi(\mathbf{q}, \mathbf{p})$ in the hierarchical ARPS approach.

To choose this matrix, as before, we have to answer two main questions regarding the desired simulation: *how* to switch the degrees of freedom on and off in the system, and *when* to do so.

5.2.1.1 How to switch

As in ARPS, we want to remove *positional* degrees of freedom in the system. However, these are not individual degrees of freedom of each particle, but *relative* positional degrees of freedom. In other words, we do not restrain and release particles independently, but consider blocks of particles being restrained *together*. For example, when two particles are merged together, they form a single rigid body. In our case, these two merged particles move together *translationally*, but no rotational motion is performed. Then, when this rigid body is split, each particle starts following its full-dynamics motion. Again, a continuous transition between these two behaviors should be provided[1].

[1] Merging and splitting of particles to simulate particle-based fluid dynamics has been used in Ref. [216]. A sampling method, that allows focusing computational resources in geometrically complex regions, has been

The problem of choosing which groups of rigid bodies should be considered for merging is combinatorial, and may thus be too slow in practice.

One possible solution is to impose constraints on pairs of rigid bodies that may be considered as candidates for merging events. For example, denoting A_1, ..., A_n a set of indexed rigid bodies, we may organize the indices into a binary tree, so that A_1 may only be merged with A_2 (when a speciifc merging condition is satisfied), A_3 may only be merged with A_4, etc., and, at higher levels in the binary tree, $(A_1 + A_2)$ may only be merged with $(A_3 + A_4)$, etc.

Another possibility would be to consider the graph deduced from proximity queries (graph nodes are particles or rigid bodies, and an edge describes the fact that the two nodes are closer than a certain cutoff distance). Then, graph edges are considered as candidates for possible "collapse", to produce a hierarchical graph which represents the hierarchy of merging operations.

In our implementation, we preferred to use a tree structure, since this allows for incremental updates of the decision metrics for the internal nodes in the hierarchy, as well as incremental updates of the corresponding partial energies and forces (corresponding algorithms will be described further). In this tree, each leaf node refers to a single particle, and each internal node represents a group of particles composed of the particles referred to by its children.

If a body C (internal tree node C) is composed of two bodies A and B (has two child nodes A and B), we define the inverse inertia matrix Φ_C corresponding to this node with a recursive formula:

$$\Phi_C = \frac{\rho_C}{m_C} E + (1 - \rho_C) \begin{bmatrix} \Phi_A & 0 \\ 0 & \Phi_B \end{bmatrix}. \tag{5.2.4}$$

In this formula, the matrix $E = \{e_{ij}\}$ is a $3n_C \times 3n_C$ matrix, where $e_{ij} = I$, $i = 1 \dots n_C, j = 1 \dots n_C$, I is a 3×3 identity matrix, n_C is the number of leaves of node C, m_C is the classical mass of this node (the sum of the masses of its leaves), and ρ_C is the *restraining function* for this node. It is clear that the inverse inertia matrix in this case is no more diagonal as in classical ARPS, and may even be dense. For the leaf nodes, this matrix Φ is set to the inverse mass of the corresponding particle multiplied by a 3×3 identity matrix.

5.2.1.2 When to switch

We consider merging two rigid bodies A and B into a rigid body C (and splitting C back into A and B) based on the restraining function ρ_C mentioned above. Precisely, an internal node C is considered to be *rigid* if ρ_C is equal to 1, and is called *non-rigid* otherwise. All tree leaves are rigid by definition (they correspond to individual particles), and remain rigid throughout the whole simulation. If ρ_C is equal to 0, node C is called a *free* node.

introduced. In our approach, we merge the *dynamics* of the particles, but preserve their geometrical details, as well as the interaction potential.

In this Chapter, we choose the restraining function in the following recursive form:

$$\rho_C = \begin{cases} 1, & \text{if } C \text{ is a leaf node,} \\ \mu(\varepsilon_C)\rho_A\rho_B, & \text{otherwise,} \end{cases} \tag{5.2.5}$$

where $\mu \in [0,1]$ is a twice-differentiable function of ε:

$$\mu(\varepsilon) = \begin{cases} 1, & \text{if } 0 \leqslant \varepsilon \leqslant \varepsilon^r, \\ 0, & \text{if } \varepsilon \geqslant \varepsilon^f, \\ s(\varepsilon) \in [0,1], & \text{elsewhere,} \end{cases} \tag{5.2.6}$$

and $s(\varepsilon)$ is a twice-differentiable function with respect to its argument and, therefore, to p_i, satisfying $s(\varepsilon_i^r) = 1$, $s(\varepsilon_i^f) = 0$. Again, as in the case of classical ARPS, a possible choice for this function is an interpolation spline of order five: $s(\varepsilon) = -6\eta^5 + 15\eta^4 - 10\eta^3 + 1$, where $\eta = (\varepsilon - \varepsilon^r)/\delta$ and $\delta = \varepsilon^f - \varepsilon^r$ is the width of the transition region. As a result, body C corresponding to an internal node can be rigid only if both of its children are rigid.

In this implementation, we choose ε_C to be dependent on the relation between the momenta of the two bodies A and B as follows[2]:

$$\begin{aligned} \varepsilon_C(p_C) = \frac{1}{2}\frac{\|p_A\|^2}{m_A} + \frac{1}{2}\frac{\|p_B\|^2}{m_B} - \frac{1}{2}\frac{\|p_A + p_B\|^2}{m_C} = \\ = \frac{1}{2}\frac{\|p_A m_B - p_B m_A\|^2}{m_A m_B m_C}, \end{aligned} \tag{5.2.7}$$

where p_C is a 3D vector, which is a sum of momenta of all leaf nodes corresponding to node C:

$$p_C = p_A + p_B = \sum_{i\in A, i=1}^{n_A} p_i + \sum_{i\in B, i=1}^{n_B} p_i = \sum_{i\in C, i=1}^{n_C} p_i,$$

where n_A, n_B and n_C correspond to the number of leaves descending from nodes A, B and C respectively. We can now define a hierarchical AR Hamiltonian:

$$H_{AR}(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\Phi_R(\mathbf{p})\mathbf{p} + V(\mathbf{q}), \tag{5.2.8}$$

where the inverse inertia matrix $\Phi_R(\mathbf{p})$ is the matrix defined by (5.2.5) for the root node of the binary tree representing the whole system of particles. The hierarchical AR Hamiltonian is separable, as this matrix only depends on particle momenta. We remind that in this case (see Section 4.2.2), correct static equilibrium properties of the system in the NVT ensemble may be computed, and the quantities that only depend on particle positions can be obtained straight away: $\langle \mathbf{A}(\mathbf{q}) \rangle_{H_{AR}} = \langle \mathbf{A}(\mathbf{q}) \rangle_H$ (an example of such quantity is the temperature of the simulation).

[2] Another criteria for merging and splitting of tree nodes in application to the computer graphics (hair motion) have been described in Ref. [217].

The equations of motion of the system in this case, can be written as follows:

$$\begin{aligned}\dot{\mathbf{p}} &= -\frac{\partial H_{AR}}{\partial \mathbf{q}} = -\frac{\partial V(\mathbf{q})}{\partial \mathbf{q}},\\ \dot{\mathbf{q}} &= \frac{\partial H_{AR}}{\partial \mathbf{p}} = \Phi_R(\mathbf{p})\mathbf{p} + \frac{1}{2}\mathbf{p}^T\frac{\partial \Phi_R(\mathbf{p})}{\partial \mathbf{p}}\mathbf{p}.\end{aligned} \tag{5.2.9}$$

It is clear from equation (5.2.4) that if node C is free ($\rho_C = 0$), then matrix Φ_C consists of two diagonal blocks corresponding to nodes A and B. In this case, equations (5.2.9) may be written separately for the subsystems corresponding to these child nodes. That is why in practice, we perform the depth-first search of the binary tree representing the system until we reach a non-rigid node C ($\rho_C \neq 0$). We then obtain the following equations for the corresponding leaves:

$$\begin{aligned}\dot{\mathbf{p}}_C &= -\frac{\partial V(\mathbf{q}_C)}{\partial \mathbf{q}_C},\\ \dot{\mathbf{q}}_C &= \Phi_C(\mathbf{p}_C)\mathbf{p}_C + \frac{1}{2}\mathbf{p}_C^T\frac{\partial \Phi_C(\mathbf{p}_C)}{\partial \mathbf{p}_C}\mathbf{p}_C,\end{aligned} \tag{5.2.10}$$

where $\mathbf{q}_C$ is the vector composed of positions of the leaf nodes corresponding to the node C and $\mathbf{p}_C$ is the vector of their momenta. The vector $\mathbf{p}_C$ combines vectors $\mathbf{p}_A$ and $\mathbf{p}_B$ of the child nodes A and B:

$$\mathbf{p}_C = \begin{bmatrix} \mathbf{p}_A \\ \mathbf{p}_B \end{bmatrix}.$$

Please note that it is possible to make a sub-system of the whole system permanently active in hierarchical AR simulations: all particles of this subsystem should be put in a separate sub-tree, the metrics ρ should be set to 0 for all internal nodes an the root node in this sub-tree, and this metrics should never be updated during the simulation. However, if the sub-system of interest changes, the whole tree might need to be rebuilt.

Please note that this approach can be easily extended for the systems represented not as a binary tree, but as a tree with more child nodes, for example, k nodes.

5.3 Discussion

We provide a simple example describing changes in the motion of the system in a hierarchical AR simulation. We considered the following system in 1D: two particles of mass 1 connected by a spring of stiffness 1. The binary tree corresponding to the system is a root node with two children corresponding to these two particles. Given some initial momenta, the two particles were oscillating in space with time, and we were observing their trajectories. The trajectories obtained with a full-dynamics simulation are shown in Fig. 5.3.1.

The trajectories of the particles obtained with hierarchical AR simulations (four different sets of thresholds were used) are shown in Fig. 5.3.2.

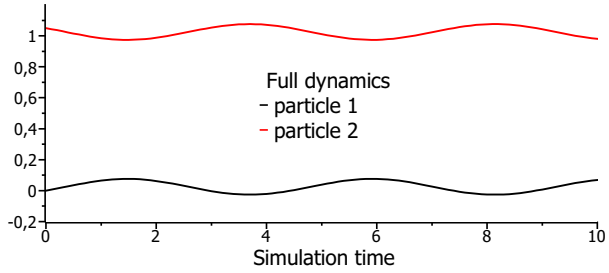


Figure 5.3.1: Trajectories in space of the two particles connected by a stiff spring in 1D during a full-dynamics simulation.

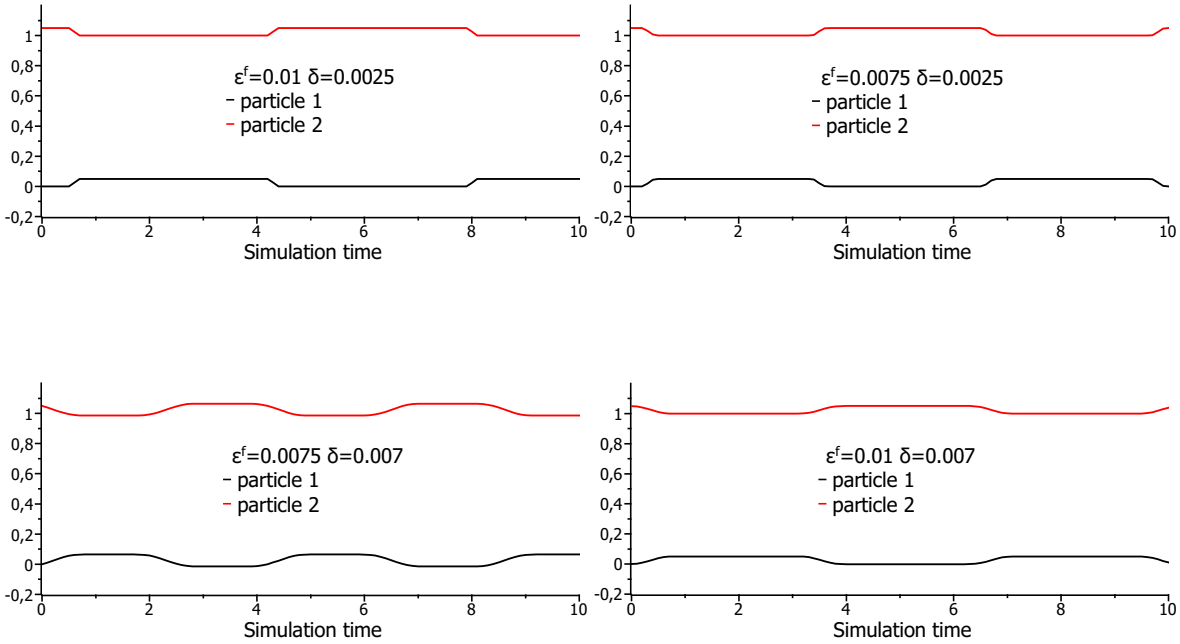


Figure 5.3.2: Trajectories of two particles connected by a stiff spring in 1D during hierarchical AR simulations with different sets of thresholds.

Algorithm 5.1 Hierarchical ARPS. General integration scheme for simulations in the NVE ensemble.

```
updateState()
    treeRoot->updateForces()
    treeRoot->updateMomenta()
    treeRoot->updateMetrics()
    treeRoot->updatePositions()
```

It can be seen from the pictures, that:

- At some moment of the simulation (in some cases, at the beginning), the momenta of the two particles become close according to the metrics (5.2.7). In general, this happens when the two particles perform close translational motions (or do not move much). Here, merging happens when the spring is "almost" stretched or "almost" compressed. As a result, the root tree node becomes rigid and both particles stop (in the general case, they would start preforming translational motion together, but in this example due to the law of preservation of momenta, they just freeze). Even when the two particles stop, though, their momenta continue evolving, breaking at some other moment the merging condition. Then, the two particles resume moving.
- For the same value of ε^f, with bigger δ, the transition region on the trajectory is wider.
- For the same value of δ, with bigger ε^f, the fully-restrained region is longer and flatter.
- For hierarchical ARPS, as for classical ARPS, the period of oscillation for the system is perturbed.

5.4 Algorithms

After having introduced and discussed hierarchical AR simulations, we can proceed to the numerical realization.

To perform a hierarchical AR simulation in the NVE ensemble, a set of equations (5.2.9) should be integrated in time. As the hierarchical AR Hamiltonian is separable, many integration schemes become symplectic and explicit as for classical ARPS.

The general scheme of the integration algorithm that we have used in our implementation for simulations in the NVE ensemble is shown in pseudo-code in Algorithm 5.1. This is a symplectic Euler scheme that is described in eq. (4.2.12). The *updateState()* function is called every time step. All functions mentioned in Algorithm 5.1 are called from the root node of the tree. We will now give more details on the functions mentioned in this algorithm.

5.4.1 Updating forces

We now describe in more detail the force update procedure.

As classical ARPS in the previous Chapter, hierarchical ARPS may speed up force computations due to the fact that interaction forces only depend on a few relative positional degrees of freedom. Under this assumption, in hierarchical ARPS, we do not need to recompute forces inside rigid bodies – *i.e.* the groups of particles in a sub-tree with a root node considered as rigid according to the decision metrics (5.2.7).

Further on we also suppose that the interaction potential is rapidly decaying (a cutoff distance is introduced).

To efficiently update forces without recomputing them inside rigid bodies, we used an algorithm proposed in Ref. [41]. This force update algorithm, relying on a binary tree representation of the system, uses a corresponding hierarchy of bounding volumes (see Chapter 2, one BV per tree node) to compute *partial interaction lists*. In our implementation, we chose to use axis-aligned bounding volumes (AABB's) because of their low update cost.

This algorithm is also using *partial force tables* – special structures that should be stored in all internal nodes. For each leaf node, a partial force table is a zero 3D vector. For each internal node C a partial force table is an array of 3D vectors, and the array's length is equal to the number of leaves descending from this internal node. Each array element corresponds to a certain leaf node. Precisely, this array element is the force acting on the leaf particle produced by all other leaf nodes of C. Each element of the root partial force table corresponds to the total force acting on a particle.

The force update procedure at each time step consists in three main steps (the potential energy can be updated at the same time).

1. The hierarchy of AABB's is updated bottom-up for the whole tree. For leaf nodes, an updated box coincides with the particle's position. The box of any internal node is obtained by bounding the boxes of its two child nodes.

2. Interaction (or neighbor) lists (see Chapter 2) are recursively constructed for as described in Ref. [76]. This means that for node C, which has child nodes A and B, the interaction list contains all pairs of particles such that one particle belongs to A and the other belongs to B. As in the rigid nodes, relative particle positions do not change, corresponding interaction lists do not change either. Therefore, these lists are computed only for *non-rigid* internal nodes of the tree. During the first time step, since only leaf nodes are rigid, we update interaction lists for all internal nodes.

3. Partial force tables are incrementally updated in a bottom-up manner. In rigid nodes, forces between particles do not change. Therefore, we update the partial force tables for non-rigid nodes only. For each non-rigid node C, we copy to the first half of its partial force table the elements of the partial force table of the node A, and to the second half, the elements of the partial force table of the node B. Then, for each element (pair of particles) of the interaction list of the node C, we calculate the interaction forces and add them to the corresponding elements of the partial force table, as if we added them to the particle's forces. Finally, forces corresponding to all particles will be stored in the partial force table of the tree root node.

Algorithm 5.2 Hierarchical ARPS. Main steps of the force-update algorithm.

updateForces()
  update AABB hierarchy:
    recursively, bottom-up, recompute boxes for every tree node;
  update interaction lists using AABB hierarchy:
    recursively, update lists for every non-rigid internal node;
  update partial force tables and potential energies:
    recursively, bottom-up, update information for every non-rigid internal node.

Algorithm 5.3 Hierarchical ARPS. Algorithm for the particle momenta update. Lower indices stand for time step numbers, the time step size is denoted by h.

updateMomenta()
  for (each particle a^i)
    $p^i_{n+1} = p^i_n + f^i_{n+1}h$

This is, however, not the optimal version of the algorithm: in the third step, copying of the child force tables to the parent one may be long and can be avoided. For more details on this method and its complexity analysis one can refer [41] and references therein.

We provide a schematic description of the this force-update method in Algorithm 5.2.

5.4.2 Updating momenta

A simple procedure of particles momenta update is performed for each particle of the system. It is described in pseudo-code in Algorithm 5.3.

5.4.3 Updating positions

We now discuss in more detail the procedure of updating particle positions.

In the general case, the complexity to evaluate at each time step the following equation:

$$\dot{\mathbf{q}} = \Phi(\mathbf{p})\mathbf{p} + \frac{1}{2}\mathbf{p}^T\frac{\partial\Phi(\mathbf{p})}{\partial\mathbf{p}}\mathbf{p}, \tag{5.4.1}$$

is cubic in the number of particles in the system ($O(N^3)$) because of the term $\dfrac{\partial\Phi(\mathbf{p})}{\partial\mathbf{p}}$. This term is a derivative of a matrix with respect to a vector, thus, a 3−dimensional object, where:

$$\frac{\partial\Phi}{\partial p_i} = \begin{bmatrix} \dfrac{\partial\Phi_{11}}{\partial p_i} & \cdots & \dfrac{\partial\Phi_{1N}}{\partial p_i} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial\Phi_{N1}}{\partial p_i} & \cdots & \dfrac{\partial\Phi_{NN}}{\partial p_i} \end{bmatrix}. \tag{5.4.2}$$

Fortunately, we can still achieve $O(N \log N)$ complexity in time, updating some special structures recursively. For example, for an internal node C with child nodes A and B we can write:

$$\begin{aligned}
&\mathbf{Q}_C = \Phi(\mathbf{p}_C)\,\mathbf{p}_C = \\
&= \left(\frac{\rho_C(p_C)}{m_C}E + (1-\rho_C(p_C))\begin{bmatrix} \Phi_A(\mathbf{p}_A) & 0 \\ 0 & \Phi_B(\mathbf{p}_B) \end{bmatrix}\right)\mathbf{p}_C = \\
&= \frac{\rho_C(p_C)}{m_C}\left\{\sum_{i\in C,\, i=1}^{n_C} p_i\right\} + (1-\rho_C(p_C))\begin{bmatrix} \Phi_A(\mathbf{p}_A)\,\mathbf{p}_A \\ \Phi_B(\mathbf{p}_B)\,\mathbf{p}_B \end{bmatrix} = \\
&= \frac{\rho_C(p_C)}{m_C}p_C + (1-\rho_C(p_C))\begin{bmatrix} \mathbf{Q}_A \\ \mathbf{Q}_B \end{bmatrix},
\end{aligned} \tag{5.4.3}$$

where$\{\sum_{i=1}^{n_C} p_i\}$ is a vector of length n_C and each component of this vector is equal to $\sum_{i=1}^{n_C} p_i$.

In a similar way, we obtain another expression for node C:

$$\begin{aligned}
&\mathbf{R}_C = \frac{\partial \rho_C(p_C)}{\partial \mathbf{p}_C} = \begin{bmatrix} \dfrac{\partial \rho_C(p_C)}{\partial \mathbf{p}_A} \\ \dfrac{\partial \rho_C(p_C)}{\partial \mathbf{p}_B} \end{bmatrix} = \{\rho_C = \mu(\varepsilon_C)\rho_A\rho_B\} = \\
&= \begin{bmatrix} \rho_B(p_B)\left(\mu(\varepsilon_C)\dfrac{\partial \rho_A(p_A)}{\partial \mathbf{p}_A} + \rho_A(p_A)\dfrac{\partial \mu(\varepsilon_C)}{\partial \varepsilon_C}\dfrac{\partial \varepsilon_C}{\partial \mathbf{p}_A}\right) \\ \rho_A(p_A)\left(\mu(\varepsilon_C)\dfrac{\partial \rho_B(p_B)}{\partial \mathbf{p}_B} + \rho_B(p_B)\dfrac{\partial \mu(\varepsilon_C)}{\partial \varepsilon_C}\dfrac{\partial \varepsilon_C}{\partial \mathbf{p}_B}\right) \end{bmatrix} = \\
&= \begin{bmatrix} \rho_B(p_B)\left(\mu(\varepsilon_C)\mathbf{R}_A + \rho_A(p_A)\dfrac{\partial \mu(\varepsilon_C)}{\partial \varepsilon_C}\left(\dfrac{p_A}{m_A} - \dfrac{p_C}{m_C}\right)\dfrac{\partial p_A}{\partial \mathbf{p}_A}\right) \\ \rho_A(p_A)\left(\mu(\varepsilon_C)\mathbf{R}_B + \rho_B(p_B)\dfrac{\partial \mu(\varepsilon_C)}{\partial \varepsilon_C}\left(\dfrac{p_B}{m_B} - \dfrac{p_C}{m_C}\right)\dfrac{\partial p_B}{\partial \mathbf{p}_B}\right) \end{bmatrix} = \\
&= \left\{p_A = \sum_{i=1}^{n_A} p_i, \quad \frac{\partial p_A}{\partial p_i} = 1\right\} = \\
&= \begin{bmatrix} \rho_B(p_B)\left(\mu(\varepsilon_C)\mathbf{R}_A + \rho_A(p_A)\dfrac{\partial \mu(\varepsilon_C)}{\partial \varepsilon_C}\left\{\dfrac{p_A}{m_A} - \dfrac{p_C}{m_C}\right\}\right) \\ \rho_A(p_A)\left(\mu(\varepsilon_C)\mathbf{R}_B + \rho_B(p_B)\dfrac{\partial \mu(\varepsilon_C)}{\partial \varepsilon_C}\left\{\dfrac{p_B}{m_B} - \dfrac{p_C}{m_C}\right\}\right) \end{bmatrix}.
\end{aligned} \tag{5.4.4}$$

Finally, we can recursively update the last expression from equation (5.4.3), using the previous equation (5.4.4):

$$\begin{aligned}
&\mathbf{S}_C = \mathbf{p}_C^T \frac{\partial \Phi_C(\mathbf{p}_C)}{\partial \mathbf{p}_C} \mathbf{p}_C = \\
&= \mathbf{p}_C^T \left(\frac{\partial}{\partial \mathbf{p}_C} \left(\rho_C(p_C) \left(\frac{1}{m_C} E - \begin{bmatrix} \Phi_A & 0 \\ 0 & \Phi_B \end{bmatrix} \right) \right) + (1 - \rho_C(p_C)) \frac{\partial}{\partial \mathbf{p}_C} \begin{bmatrix} \Phi_A & 0 \\ 0 & \Phi_B \end{bmatrix} \right) \mathbf{p}_C = \\
&= \mathbf{p}_C^T \left(\frac{\partial}{\partial \mathbf{p}_C} \left(\rho_C(p_C) \left(\frac{1}{m_C} E - \begin{bmatrix} \Phi_A & 0 \\ 0 & \Phi_B \end{bmatrix} \right) \right) \right) \mathbf{p}_C + \\
&\qquad + (1 - \rho_C(p_C)) \mathbf{p}_C^T \left(\frac{\partial}{\partial \mathbf{p}_C} \begin{bmatrix} \Phi_A & 0 \\ 0 & \Phi_B \end{bmatrix} \right) \mathbf{p}_C = \\
&= \frac{\partial \rho(p_C)}{\partial \mathbf{p}_C} \left(\mathbf{p}_C^T \left(\frac{1}{m_C} E - \begin{bmatrix} \Phi_A & 0 \\ 0 & \Phi_B \end{bmatrix} \right) \mathbf{p}_C \right) + (1 - \rho_C(p_C)) \begin{bmatrix} \mathbf{p}_A^T \frac{\partial \Phi(\mathbf{p}_A)}{\partial \mathbf{p}_A} \mathbf{p}_A \\ \mathbf{p}_B^T \frac{\partial \Phi(\mathbf{p}_B)}{\partial \mathbf{p}_B} \mathbf{p}_B \end{bmatrix} = \\
&= \frac{\partial \rho_C(p_C)}{\partial \mathbf{p}_C} \left(\mathbf{p}_C \cdot \left(\frac{1}{m_C} \left\{ \sum_{i \in C, i=1}^{n_C} p_i \right\} - \begin{bmatrix} \mathbf{Q}_A \\ \mathbf{Q}_B \end{bmatrix} \right) \right) + (1 - \rho_C(p_C)) \begin{bmatrix} \mathbf{S}_A \\ \mathbf{S}_B \end{bmatrix} = \\
&= \mathbf{R}_C \left(\mathbf{p}_C \cdot \left(\frac{p_C}{m_C} - \begin{bmatrix} \mathbf{Q}_A \\ \mathbf{Q}_B \end{bmatrix} \right) \right) + (1 - \rho_C(p_C)) \begin{bmatrix} \mathbf{S}_A \\ \mathbf{S}_B \end{bmatrix},
\end{aligned} \tag{5.4.5}$$

where the dot sign stands for the dot product.

In Algorithm 5.4 we describe in pseudo-code functions that modify structures $\mathbf{Q}_C$, $\mathbf{R}_C$ and $\mathbf{S}_C$: *init()* sets them up at the beginning of the simulation, *updateMetrics()* updates them according to equations (5.4.3-5.4.5) and should be called before updating positions.

In Algorithm 5.5 we describe in pseudo-code the function updating particles positions.

The space complexity of updating positions is, therefore, $O(N \log N)$: for each internal tree node we store four vectors of length equal to the number of underlying leaves. These vectors are $\mathbf{Q}_C$, $\mathbf{R}_C$, $\mathbf{S}_C$ and a partial force table. The time complexity is also $O(N \log N)$ since these structures need to be updated at each time step, and Q_C is always updated for all nodes, including leaves.

We remind that the integration scheme for the simulations in the NVT ensemble described in Section 4.2.3 may be also used for hierarchical ARPS.

Algorithm 5.4 Hierarchical ARPS. Updating structures $\mathbf{Q}_C$, $\mathbf{R}_C$ and $\mathbf{S}_C$. Each internal node C is considered to have two child nodes A and B.

init()
- **for** (each internal node C)
  - $\rho_C = 0$, $\mathbf{Q}_C$.setZero(), $\mathbf{R}_C$.setZero(), $\mathbf{S}_C$.setZero()
- **for** (each leaf node)
  - $\rho = 1$, $Q = p/m$, $R = 0$, $S = 0$

updateMetrics()
- **if** (this node C is internal)
  - A->updateMetrics()
  - B->updateMetrics()
  - update ε_C (eq. 5.2.7)
  - updateρ_C: $\rho_C = \mu(\varepsilon_C)\rho_A\rho_B$
  - if ($\rho_C > 0$) [a]
    - update $\mathbf{Q}_C$ (eq. 5.4.3), $\mathbf{R}_C$ (eq. 5.4.4), $\mathbf{S}_C$ (eq. 5.4.5)
- **else** // this node is a leaf corresponding to a particle
  - $Q = p/m$

[a] This is an optimization: since we do not use any metrics for free nodes, we do not update the metrics here.

Algorithm 5.5 Hierarchical ARPS. Updating positions. Each internal node C is considered to have two child nodes A and B. Lower indices stand for time step, upper indices are particles indices, q denotes particles positions, h denotes the time step size.

updatePositions()
- **if** ($\rho_C = 0$)
  - // node C is a free node
  - A->updatePositions()
  - B->updatePositions()
- **else** // node C
  - **for** (each leaf l^i)
    - $q^i_{n+1} = q^i_n + (\mathbf{Q}_C[i] + 0.5\mathbf{S}_C[i])h$

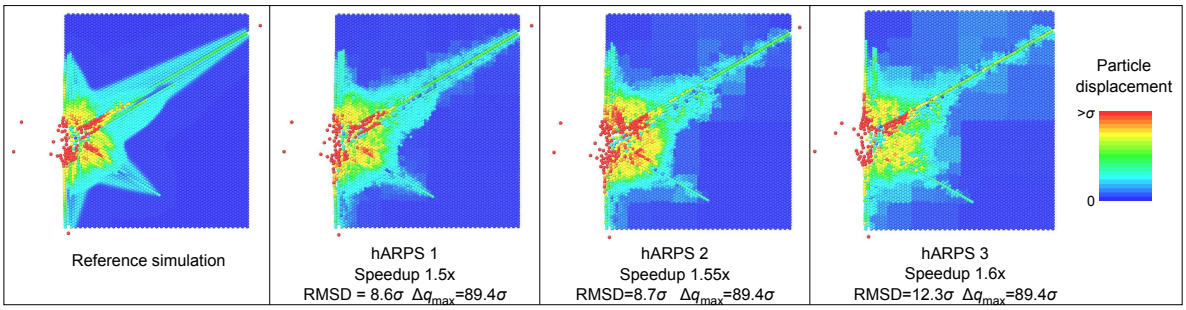


Figure 5.5.1: Hierarchical ARPS. Simulating a collision cascade with controlled precision. Hierarchical adaptively restrained simulations allow us to smoothly trade between precision and speed. The main features of the shock are preserved. The binary tree representation was constructed top-down.

5.5 Results

The hierarchical ARPS method was implemented in C++ and tested on an Intel 2.40 GHz quad-core processor with 4GB of RAM, on a Windows Vista 32-bit operating system.

In this section, we illustrate the hierarchical ARPS approach with several numerical examples, designed for molecular dynamics simulations.

5.5.1 Collision cascade in 2D

We first simulated a collision cascade in a 2D system composed of 5 930 particles with mass 1 g/mol in the NVE ensemble, using Lennard-Jones potential ($\epsilon/k_B = 120$ K, $\sigma = 3.4$ Å, cutoff = 8 Å, the potential was truncated through a smoothing function applied between 7.5 and 8 Å). We performed four simulations of the shock induced by a particle launched at high velocity towards the initially static 2D system: a reference (full-dynamics) simulation, and three hierarchical AR simulations with varying degrees of precision (time step size 0.0488 fs, 7 000 time steps, total simulation time 342 fs). Figure 5.5.1 compares the final configurations reached by the four simulations. For each hierarchical AR simulation, the Root-Mean-Square Deviation (RMSD) from the reference final configuration is given, as well as the maximum particle displacement error $\Delta q_{\max}$.

The parameters of the hierarchical AR simulations that we launched in this test were the following:

- hARPS 1: $\varepsilon^r = 0.01$ kcal/mol, $\varepsilon^f = 1.01$ kcal/mol ($\delta = 1$ kcal/mol);
- hARPS 2: $\varepsilon^r = 1$ kcal/mol, $\varepsilon^f = 2$ kcal/mol ($\delta = 1$ kcal/mol);
- hARPS 3: $\varepsilon^r = 3$ kcal/mol, $\varepsilon^f = 4$ kcal/mol ($\delta = 1$ kcal/mol).

For hierarchical AR simulations, the results depend on the tree representation of the system. For example, for the results demonstrated in Fig. 5.5.1 the tree was constructed in

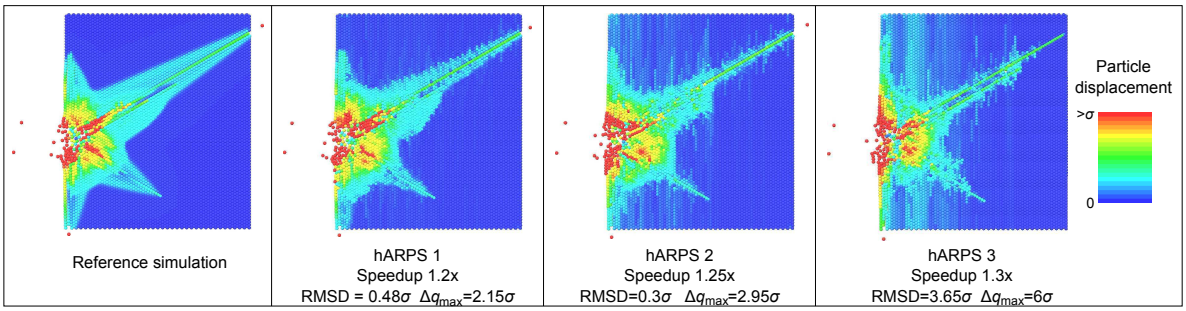


Figure 5.5.2: Hierarchical ARPS. Simulating a collision cascade with controlled precision. Hierarchical adaptively restrained simulations allow us to smoothly trade between precision and speed. The main features of the shock are preserved. The binary tree representation was constructed bottom-up.

a top-down manner by recursive dividing of the system in halves as described in 2.2.2.1 and, therefore, the squares of different levels are being activated by the shock.

To clearly demonstrate the effect of the tree, we provide the results for the same four simulations with another tree built in a bottom-up manner by grouping the particles pairwise according to their sequence number (they were enumerated, first, along the y-axis, vertically, and then, along the x-axis, horizontally). These results are shown in Fig. 5.5.2, and are rather different from those in Fig. 5.5.1: vertical lines are being activated when the central part of the plane is reached by the shock.

Speedup for the hierarchical AR simulations in this study is not big and this is relative to the reference simulation, which was an AR simulation with both thresholds ε^f and ε^r equal to zero (still performing the update of AABB-hierarchy, etc). The real full-dynamics simulation, just following equations (5.2.2) is more rapid than our reference simulation (about 2 times faster for this simulation). Note, however, that we are not using the most optimal version of the force update algorithm.

We do not achieve significant speedups in this example because this is not the best case for the hierarchical ARPS: a lot of particles become active, making big sub-trees active as well. This is not the system requiring the hierarchical representation.

5.5.2 Radial distribution function

We, then, demonstrate the ability of hierarchical ARPS to collect unbiased statistics in the NVT ensemble. For that, as in Chapter 4, we performed Langevin simulations of a system of 343 Argon particles interacting in a 3D periodic box through Lennard-Jones potential with the same parameters as in the previous example (box length 25.56 Å, Langevin friction coefficient $\gamma = 1$, $T = 94.4$ K). The particles were initially positioned at the nodes of a 3D cubical lattice, far from equilibrium. We launched a full-dynamics simulation and a hierarchical AR simulation (180 rigid bodies on average at each time step, $\varepsilon^r = 1$ kcal/mol and $\varepsilon^f = 2$

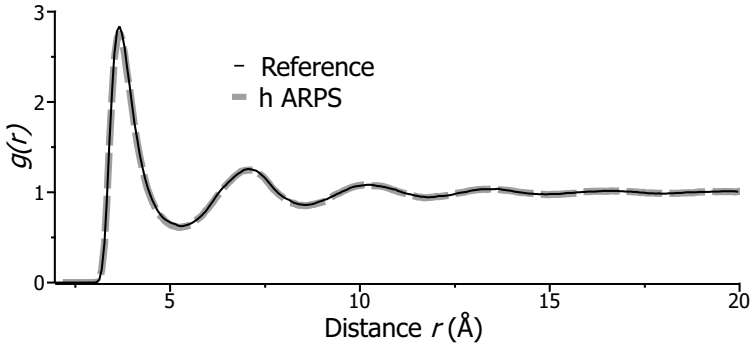


Figure 5.5.3: Hierarchical ARPS. Radial distribution functions for an Argon system (periodic 3D box, 343 particles, NVT ensemble). The curves corresponding to the reference and the AR simulations coincide.

kcal/mol) and computed the Radial Distribution Function (RDF) $g(r)$ in each case (time step size 0.488 fs, 150 000 time steps, total simulation length 73.2 ps). Figure 5.5.3 plots the results: the curves coincide, demonstrating that the equilibrium statistics have been preserved.

5.5.3 Discussion on stability

From the numerical experiments that we performed, we concluded that the hierarchical ARPS method is less stable than the classical ARPS method. For example, we noticed that, to obtain a NVE simulation with similar energy oscillations, we had to choose a smaller time step in hierarchical AR simulations.

We believe this happens because when a lot of particles are merged into a single rigid body and this body moves, the larger the body, the more it perturbs the energy of the system. A solution to this instability problem might be to introduce softer potentials for larger rigid bodies. However, as it was discussed before, the parametrization of a potential function is not an easy task.

5.6 Conclusion

In this Chapter, we have introduced an alternative choice of the inverse inertia matrix in an adaptively restrained Hamiltonian, different from the one described in the previous Chapter. The inverse inertia matrix that we have proposed is designed for systems represented as a binary hierarchy, and we call the corresponding approach hierarchical Adaptively Restrained Particle Simulations.

Hierarchical ARPS may be useful to simulate systems for which a hierarchical representation is natural (such as articulated-body systems, although we will need a more sophisticated inverse inertia, that will depend on positions, since this is already the case in non-adaptive articulated-body dynamics). Simulations that do not require frequent tree rebuilding are also preferable.

In the future, it might be interesting to study coarse-graining in the traditional sense with this method: to fix a specific coarse-graining level, *e.g.*, water molecules, and, therefore, the corresponding level in a tree. The advantage of our method is that we can choose any coarse-graining level without changing the potential function.

Chapter 6

Conclusions and outlook

To conclude this disseration, we now briefly describe its main contributions and provide several possible directions of the future research.

In this thesis, we have reviewed numerous algorithms designed to perform efficient molecular simulations. Being widely used in many research domains, including *e.g.* computational chemistry, physics and material science, molecular simulations still need to be accelerated to help us to better understand chemical reactions and various processes at the atomistic level, and to solve many challenging problems, for example protein folding or molecular docking. This work contains three main contributions aiming at making molecular simulations more efficient. The first one compares various methods for neighbor list construction, while the two other contributions describe novel algorithms. While the first two contributions concern only molecular simulations, and, moreover, specific types of such simulations, the last part of this thesis (its major contribution) was designed for particle simulations, a larger class of simulations having application in numerous research fields.

The first part of this dissertation is devoted to accelerating an important part of a molecular simulation: neighbor list construction for the force computation. Precisely, we have proposed to apply to molecular simulations an algorithm that is typically used in computational geometry and its applications (robotics, virtual reality, etc.). We have compared several modifications of this hierarchy-based algorithm to the algorithm classically used in molecular simulations and relying on a grid. We showed that, for large rigid molecules interacting through a small contact area, as in *e.g.* rigid-body docking, the hierarchy-based approach is beneficial.

Several directions might thus be interesting to explore in the future:

- The two neighbor search methods, the hierarchical approach and the grid-based method, might be combined as follows. First, a hierarchical approach might be used to eliminate large irrelevant groups of atoms. Then, a grid method should locate neighboring pairs in the refined search region. The study on the size of the former region should also be performed.

- The hierarchy-based algorithm may be combined with the method described in Ref. [166]. The latter algorithm finds interactions between two groups of particles, avoiding unnecessary distance computations, and might be used to inspect the particles of the two leaf hierarchy nodes.

- Shortly after our paper [127] was published, a new method has been proposed for neighbor search for large rigid molecules: a rigid cell linked-list algorithm [218]. It would be interesting to compare it to the hierarchy-based approach.

In the second part of this thesis, we have presented a fast algorithm for building a binary hierarchy associated to a molecular graph that may contain cycles and multiple connected components. This algorithm consists in three steps. The first and the second steps are based on the system's topology, and result in a binary tree for each graph connected component. The third step relies on the system's geometry, and gathers the trees corresponding to all connected components into a final tree. We applied this algorithm to systems simulated in torsion-angle coordinates for adaptive quasi-statics of biomolecules.

The tree-construction algorithm mainly concerns the offline part of the simulation: its initialization. However, when the tree needs to be recomputed several times during the simulation, its fast re-building or update can play an important role. This recomputing may be necessary to better reflect the geometrical properties of the system: for example, when the relative positions of the molecules in the system change.

The following possible research directions should be indicated:

- A study of several extensions, including *e.g.* dynamic tree updates after insertions in, or deletions from, the molecular graph, or when the graph's topology has been changed.

- The binary tree for the hierarchy-based method described in Chapter 2 might be constructed with the third step of the proposed algorithm. The quality of the trees and, therefore, the speed of obtaining neighbor lists should be compared.

- More applications of the tree construction algorithm (those relying on hierarchical representations or those using torsion-angle dynamics) should be studied in other research domains as, *e.g.* robotics or computer graphics.

In the third part of this work (Chapters 4 and 5), we have introduced a novel general approach to particle (and, in particular, molecular) simulations that we call Adaptively Restrained Particle Simulations. This approach accelerates dynamics of the particle system by adaptively switching positional degrees of freedom on and off during a simulation. This switching is done with the help of a modified inverse inertia matrix in the original Hamiltonian. As a result, for most potentials, a smaller number of forces may be computed at each time step.

We described two possible choices of the inverse inertia matrix for simulations in Cartesian coordinates. In the first case, each particle is restrained and released independently, according to its kinetic energy. In the second case, particles are organized in a binary tree, and can

be merged together and released according to this structure when a condition based on their relative momenta is satisfied.

This method has numerous advantages and we demonstrated on several examples its ability to produce long stable simulations and efficiently compute statistics.

The work on this very general approach should be continued. The fact that the inverse inertia matrix can be chosen according to the needs of a specific problem opens a way to a large number of applications in various research fields. That brings us to numerous possible directions of the future research:

- New applications should be examined, and new specific inverse inertia matrices should be constructed. Among these applications might be crack propagation, fracture in metals [208], ion implantation, channels in membrane proteins, molecular docking, protein folding, molecular solvation, conformational sampling, free energy computations, etc. In general, we believe that this method will be beneficial for the simulations, where the statistics are computed only for a local part of the molecular system.

- An inverse inertia matrix imposing an "active region" on the system may be developed: in this region of the highest interest in the system, the particles will never be restrained. The properties of the produced simulations should be studied.

- ARPS should be applied to well-known interaction potentials (CHARMM, AMBER, GROMOS, etc.), and larger problems.

- Theoretical properties of ARPS should be studied in more detail: for example, dynamical properties of the simulated system, the choice of the threshold parameters and the influence of their difference (the width δ) on the stability of the simulation, etc.

- An analytical solution for some parts of the AR simulations should be obtained. For example, it might be possible to compute in advance the time at which some particles will be released (in particular when the force applied to these particles is constant), and not update their momenta in between. This might change the computational complexity of the approach, and may result in faster time steps.

- The results of AR simulations in the polymer-in-solvent study should be compared to existing implicit methods, as this is done in Ref. [98].

- An interesting study might be to combine hierarchical ARPS with the tree construction algorithm from Chapter 3: testing ARPS on a tree built by the GofAsTr library, and comparing the results to those obtained with an arbitrarily constructed tree.

- The method should be combined with other existing methods, such as fast algorithms to compute long-range interactions [74, 75] (although incremental versions may have to be designed), as well as techniques aimed at accelerating sampling [97, 85, 98].

- We should test ARPS for free energy computations, still a significant challenge in many cases.

Software availability

The software will be made available through SAMSON (Software for Adaptive Modeling and Simulation Of Nanosystems), developed in the NANO-D group (http://nano-d.inrialpes.fr/).

Chapter 6

Conclusions et perspectives (Français)

Dans ce Chapitre nous présentons les conclusions sur les travaux effectués, et suggérons quelques travaux futurs.

Dans cette thèse, nous avons examiné de nombreux algorithmes créés pour effectuer des simulations moléculaires efficaces. Les simulations moléculaires sont largement utilisées dans de nombreux domaines de recherche, y compris la chimie et la physique numériques ainsi que la science des matériaux. Pour résoudre beaucoup de problèmes difficiles dans ces domaines leurs performances doivent être encore améliorées. Ce travail contient trois contributions principales visant à rendre les simulations moléculaires plus efficaces. La première contribution consiste à comparer les différentes méthodes de construction de la liste des voisins, tandis que les deux autres décrivent de nouveaux algorithmes. Alors que les deux premières contributions ne concernent que les simulations moléculaires, la dernière partie de la thèse (sa plus grande contribution) peut être utilisée pour une classe plus large de simulations, appliquée dans divers domaines de recherche : les simulations de particules.

La première partie de la thèse est consacrée à l'accélération d'une partie importante des simulations moléculaires qui est la construction de la liste des voisins pour le calcul des forces. Précisément, nous avons proposé d'appliquer aux simulations moléculaires un algorithme fondé sur la représentation hiérarchique. Cet algorithme est typiquement utilisé dans la géométrie numérique et ses applications (robotique, réalité virtuelle, etc.) Nous avons comparé plusieurs modifications de cet algorithme à celui, basé sur une grille, qui est traditionnellement utilisé dans les simulations moléculaires. Nous avons démontré que, pour les molécules rigides et relativement grandes interagissant à travers une petite surface de contact (c'est le cas par exemple de l'assemblage des corps rigides), l'approche basée sur la hiérarchie est plus avantageuse.

Plusieurs directions pourraient être intéressantes à explorer dans le futur :

- Les deux méthodes de recherche des voisins, l'approche hiérarchique et la méthode basée sur une grille, pourraient être combinées comme suit. D'abord, une approche hiérarchique pourrait être utilisée pour éliminer les grands groupes pertinents d'atomes.

Ensuite, une méthode avec grille devrait retrouver les paires voisines dans la zone de recherche raffinée.

- L'algorithme basé sur la hiérarchie peut être combiné avec la méthode décrite dans la Réf. [166].
- Peu de temps après que le papier [127] ait été publié, une nouvelle méthode a été proposée pour la recherche des voisins dans les grandes molécules rigides [218]. Il serait intéressant de la comparer à l'approche hiérarchique.

Dans la deuxième partie de cette thèse, nous avons proposé un algorithme rapide pour construire une hiérarchie binaire, associé à un graphe moléculaire pouvant contenir des cycles et plusieurs composantes connexes. Cet algorithme consiste en trois étapes. La première et la deuxième étapes sont basées sur la topologie du système, et produisent un arbre binaire pour chaque composante connexe. La troisième étape utilise la géométrie du système, et rassemble dans un arbre final les arbres correspondant à toutes les composantes connexes. Nous avons appliqué cet algorithme aux simulations en angles de torsion.

L'algorithme de construction d'arbre concerne principalement la partie "hors ligne" de la simulation : l'initialisation. Toutefois, lorsque l'arbre a besoin d'être recalculé plusieurs fois au cours de la simulation, la rapidité de sa reconstruction ou sa mise à jour peut jouer un rôle important. Cette reconstruction peut être nécessaire afin de mieux refléter les propriétés géométriques du système : par exemple, lorsque les positions relatives des molécules du système ont été modifiées de manière importante.

Les travaux futurs sont par exemple :

- Plusieurs extensions d'algorithme peuvent être considérées, par exemple la mise à jour dynamique de l'arbre après insertion ou suppression d'un sommet du graphe moléculaire, ou lorsque la topologie du graphe a été changée.
- L'arbre binaire de la méthode hiérarchique du Chapitre 2 pourrait être construit avec la troisième étape de l'algorithme proposé.
- D'autres applications potentielles de l'algorithme devraient être étudiées dans d'autres domaines de recherche, par exemple la robotique ou l'infographie.

Dans la troisième partie de ce travail (Chapitres 4 et 5), nous avons introduit une nouvelle approche générale pour les simulations de particules (et, notamment, des simulations moléculaires) que nous avons appelé ARPS: Simulations de Particules Restreintes de façon Adaptative (*ARPS: Adaptively Restrained Particle Simulations*). Cette approche active et désactive les degrés de liberté en position du système lors d'une simulation, grâce à une modification de la matrice d'inertie inverse dans le hamiltonien original. En conséquence, pour la plupart des potentiels, moins de forces doivent être calculées à chaque pas de temps et la dynamique du système de particules est accélérée.

Nous avons décrit deux choix possibles de la matrice d'inertie inverse pour les simulations en coordonnées cartésiennes. Dans le premier cas, chaque particule est indépendamment figée

et libérée dans l'espace, en fonction de son énergie cinétique. Dans le second cas, les particules sont organisées dans un arbre binaire, et peuvent être fusionnées ensemble ou libérées selon la structure de l'arbre, lorsqu'une condition sur leurs moments relatives est satisfaite.

Cette méthode présente de nombreux avantages et nous avons démontré sur plusieurs exemples sa capacité à produire des simulations longues et stables ainsi qu'à collecter les propriétés statiques d'équilibre correctement.

Le travail sur cette approche très générale doit être poursuivi. Le fait que la matrice d'inertie inverse puisse être adaptée à chaque problème spécifique ouvre la voie à un grand nombre d'applications dans divers domaines de recherche. Cela nous amène à de nombreux axes de recherche possibles :

- De nouvelles applications d'ARPS devraient être examinées : par exemple la rupture dans les métaux [208], l'implantation ionique, l'assemblage moléculaire, le repliement des protéines, la solvatation moléculaire, les calculs d'énergie libre, etc. De nouvelles matrices d'inertie inverse devraient être construites pour cela.

- Une matrice d'inertie inverse qui impose une "région active" sur le système peut être introduite : dans cette région de plus haut intérêt du système, des particules ne seraient jamais fixées. Les propriétés des simulations obtenues par ce biais doivent être étudiées.

- La méthode doit être appliquée à des potentiels d'interaction connus (*e.g.* CHARMM, AMBER, Gromos, etc).

- Les propriétés théoriques de ARPS devraient être étudiées plus en détail : par exemple, le choix des paramètres de seuil, l'influence de leur différence sur la stabilité de la simulation. Les propriétés dynamiques du système simulé doivent également être examinées.

- Les solutions analytiques pour certaines parties des simulations AR doivent être obtenues. Par exemple, il pourrait être possible de calculer à l'avance le moment auquel certaines particules seraient libérées. Cela pourrait accélérer les calculs.

- Les résultats de l'étude du polymère solvaté doivent être comparés à des méthodes implicites existantes, comme cela se fait dans la Réf. [98].

- Il pourrait être intéressant de combiner ARPS hiérarchiques avec l'algorithme de construction de l'arbre de Chapitre 3.

- La méthode doit être combinée avec d'autres méthodes existantes, telles que des algorithmes rapides pour calculer les interactions à longue portée [74, 75] (bien que des versions incrémentales puissent être nécessaires), ainsi que des techniques accélérant l'échantillonnage [97, 85, 98].

- Nous devrions appliquer ARPS aux calculs d'énergie libre.

Disponibilité

Les méthodes développées dans cette thèse seront mises à disposition via SAMSON (Système Adaptatif pour la Modélisation et la Simulation d'Objets Nanoscopiques — Software for Adaptive Modeling and Simulation Of Nanosystems), développé dans l'equipe NANO-D (http://nano-d.inrialpes.fr/).

Bibliography

[1] The PyMOL Molecular Graphics System, Version 1.2r3pre, Schrödinger, LLC. xviii, 12, 25, 41, 43

[2] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996. xviii, 12, 25, 42, 43, 77

[3] M.P. Allen and D.J. Tildesley. *Computer simulation of liquids*. Oxford University Press, New York, 1989. 1, 15

[4] A. Gavezzotti. *Molecular aggregation: structure analysis and molecular simulation of crystals and liquids*. Oxford University Press, USA, 2007. 1, 15

[5] S. Jones and J.M. Thornton. Principles of protein-protein interactions. *Proceedings of the National Academy of Sciences of the United States of America*, 93(1):13–20, 1996. 1, 15, 31, 89

[6] L.R. Forrest and M.S.P. Sansom. Membrane simulations: bigger and better? *Current opinion in structural biology*, 10(2):174–181, 2000. 1, 15

[7] I.Y. Gushchin, V.I. Gordeliy, and S. Grudinin. Role of the HAMP domain region of sensory rhodopsin transducers in signal transduction. *Biochemistry*, 50(4):574–580, 2010. 1, 15

[8] V. Galiatsatos. *Molecular simulation methods for predicting polymer properties*. Wiley-Interscience, 2005. 1, 15

[9] T. Schlick. *Molecular modeling and simulation: an interdisciplinary guide*. Springer Verlag, 2010. 1, 15

[10] P. Paricaud, M. Předota, A.A. Chialvo, and P.T. Cummings. From dimer to condensed phases at extreme conditions: Accurate predictions of the properties of water by a Gaussian charge polarizable model. *The Journal of Chemical Physics*, 122(24):244511, 2005. 1, 15

[11] F. Ooms. Molecular modeling and computer aided drug design. Examples of their applications in medicinal chemistry. *Current medicinal chemistry*, 7(2):141–158, 2000. 1, 15

[12] C.A. Taft, V.B. da Silva, and C.H.T.P. da Silva. Current topics in computer-aided drug design. *Journal of Pharmaceutical Sciences*, 97(3):1089–1098, 2008. 1, 15

[13] A. Wlodawer and J. Vondrasek. INHIBITORS OF HIV-1 PROTEASE: A Major Success of Structure-Assisted Drug Design 1. *Annual review of biophysics and biomolecular structure*, 27(1):249–284, 1998. 1, 15

[14] H. Alonso, A.A. Bliznyuk, and J.E. Gready. Combining docking and molecular dynamic simulations in drug design. *Medicinal research reviews*, 26(5):531–568, 2006. 1, 15

[15] G.A. Mansoori. *Principles of nanotechnology: molecular-based study of condensed matter in small systems*. World Scientific Publishing Company Incorporated, 2005. 1, 15

[16] G. Cuniberti, G. Fagas, and K. Richter. *Introducing molecular electronics*. Springer, 2005. 1, 15

[17] C.B. Anfinsen. Studies on the principles that govern the folding of protein chains. *Science*, 181(4096):223–230, 1973. 1, 15, 89

[18] K.M. Merz, M. Scott, and L. Grand. *The Protein Folding Problem and Tertiary Structure Prediction*. Birkhäuser, 1994. 1, 15, 89

[19] A.R. Leach. *Molecular modelling*. Longman Singapore, 1996. 1, 11, 15, 24, 89

[20] R. MacKinnon. Potassium channels. *FEBS letters*, 555(1):62–65, 2003. 1, 15, 89

[21] R. Featherstone. A divide-and-conquer articulated-body algorithm for parallel O(log (n)) calculation of rigid-body dynamics. Part 1: Basic algorithm. *The International Journal of Robotics Research*, 18(9):867–875, 1999. 2, 3, 7, 11, 16, 17, 20, 25, 59, 82, 83

[22] R. Featherstone. A divide-and-conquer articulated-body algorithm for parallel O(log (n)) calculation of rigid-body dynamics. Part 2: Trees, loops, and accuracy. *The International Journal of Robotics Research*, 18(9):876–892, 1999. 3, 7, 17, 20, 59, 65, 82, 83

[23] D. Frenkel and B. Smit. *Understanding molecular simulation: from algorithms to applications*. Academic Press, 2002. 3, 4, 17, 31, 32, 90, 92, 108

[24] G.J. Martyna, M.L. Klein, and M. Tuckerman. Nose-Hoover chains: The canonical ensemble via continuous dynamics. *Journal of Chemical Physics*, 97(4):2635, 1992. 3, 17

[25] A.Z. Panagiotopoulos. Direct determination of phase coexistence properties of fluids by Monte Carlo simulation in a new ensemble. *Molecular Physics*, 61(4):813–826, 1987. 4, 5, 17, 19

[26] J.W. Ponder and D.A. Case. Force fields for protein simulations. *Advances in protein chemistry*, 66:27–85, 2003. 4, 18

[27] S. Chib and E. Greenberg. Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335, 1995. 5, 19

[28] J.S. Liu, F. Liang, and W.H. Wong. The multiple-try method and local optimization in Metropolis sampling. *Journal of the American Statistical Association*, 95(449):121–134, 2000. 5, 19

[29] J.I. Siepmann and D. Frenkel. Configurational bias Monte Carlo: a new sampling scheme for flexible chains. *Molecular Physics*, 75(1):59–70, 1992. 5, 10, 19, 24

[30] J.P. Ryckaert, G. Ciccotti, and H.J.C. Berendsen. Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *Journal of Computational Physics*, 23(3):327–341, 1977. 6, 10, 19, 24

[31] H.C. Andersen. RATTLE: A 'Velocity' version of the SHAKE algorithm for molecular dynamics calculations. *Journal of Computational Physics*, 52(1):24–34, 1983. 6, 10, 19, 24

[32] A.C.T. Van Duin, S. Dasgupta, F. Lorant, and W.A. Goddard III. ReaxFF: a reactive force field for hydrocarbons. *The Journal of Physical Chemistry A*, 105(41):9396–9409, 2001. 6, 19

[33] D.W. Brenner. Empirical potential for hydrocarbons for use in simulating the chemical vapor deposition of diamond films. *Physical Review B*, 42(15):9458, 1990. 6, 19, 20, 91

[34] A.K. Rappe, C.J. Casewit, K.S. Colwell, W.A. Goddard III, and W.M. Skiff. UFF, a full periodic table force field for molecular mechanics and molecular dynamics simulations. *Journal of the American Chemical Society*, 114(25):10024–10035, 1992. 6, 20

[35] P.V.R Schleyer, A.D. MacKerell, J.R.B. Brooks, C.L. Brooks III, L. Nilsson, B. Roux, Y. Won, and M. Karplus. CHARMM: The Energy Function and Its Parameterization with an Overview of the Program. *The Encyclopedia of Computational Chemistry*, 1:271–277, 1998. 6, 20, 91

[36] H. Lin and D.G. Truhlar. QM/MM: what have we learned, where are we, and where do we go from here? *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta)*, 117(2):185–199, 2007. 6, 20

[37] L. Yang, C. Tan, M.J. Hsieh, J. Wang, Y. Duan, P. Cieplak, J. Caldwell, P.A. Kollman, and R. Luo. New-generation amber united-atom force field. *The Journal of Physical Chemistry B*, 110(26):13166–13176, 2006. 6, 20

[38] W.L. Jorgensen, J.D. Madura, and C.J. Swenson. Optimized intermolecular potential functions for liquid hydrocarbons. *Journal of the American Chemical Society*, 106(22):6638–6646, 1984. 6, 20

[39] M.G. Martin and J.I. Siepmann. Transferable potentials for phase equilibria. 1. United-atom description of n-alkanes. *The Journal of Physical Chemistry B*, 102(14):2569–2577, 1998. 6, 20

[40] A. Jain, N. Vaidehi, and G. Rodriguez. A fast recursive algorithm for molecular dynamics simulation. *Journal of Computational Physics*, 106(2):258–268, 1993. 6, 20, 59, 80, 82

[41] R. Rossi, M. Isorce, S. Morin, J. Flocard, K. Arumugam, S. Crouzy, M. Vivaudou, and S. Redon. Adaptive torsion-angle quasi-statics: a general simulation method with applications to protein structure analysis and design. *Bioinformatics*, 23(13):i408, 2007. 6, 8, 13, 20, 22, 27, 59, 80, 82, 83, 86, 140, 141

[42] D.S. Bae and E.J. Haug. A recursive formulation for constrained mechanical system dynamics: Part I. Open loop systems. *Journal of Structural Mechanics*, 15(3):359–382, 1987. 6, 20, 59, 80

[43] P. Güntert, C. Mumenthaler, and K. Wüthrich. Torsion angle dynamics for NMR structure calculation with the new program D1. *Journal of Molecular Biology*, 273(1):283–298, 1997. 6, 7, 20, 80

[44] R. Abagyan, M. Totrov, and D. Kuznetsov. ICM – a new method for protein modeling and design: applications to docking and structure prediction from the distorted native conformation. *Journal of Computational Chemistry*, 15(5):488–506, 1994. 6, 7, 20, 80

[45] A.K. Mazur and R.A. Abagyan. New methodology for computer-aided modelling of biomolecular structure and dynamics. 1. Non-cyclic structures. *Journal of Biomolecular Structure & Dynamics*, 6(4):815–832, 1989. 6, 20, 80

[46] A.K. Mazur, V.E. Dorofeev, and R.A. Abagyan. Derivation and testing of explicit equations of motion for polymers described by internal coordinates. *Journal of Computational Physics*, 92(2):261–272, 1991. 6, 7, 20, 80

[47] S. He and H.A. Scheraga. Macromolecular conformational dynamics in torsional angle space. *The Journal of Chemical Physics*, 108:271–286, 1998. 6, 20, 80

[48] V. Katritch, M. Totrov, and R. Abagyan. ICFF: A new method to incorporate implicit flexibility into an internal coordinate force field. *Journal of Computational Chemistry*, 24(2):254–265, 2003. 7, 20

[49] T. Schlick, E. Barth, and M. Mandziuk. Biomolecular dynamics at long timesteps: Bridging the timescale gap between simulation and experimentation. *Annual review of biophysics and biomolecular structure*, 26(1):181–222, 1997. 7, 20

[50] S.J. Marrink, H.J. Risselada, S. Yefimov, D.P. Tieleman, and A.H. De Vries. The MARTINI force field: coarse grained model for biomolecular simulations. *The Journal of Physical Chemistry B*, 111(27):7812–7824, 2007. 7, 21, 91

[51] G.A. Voth, editor. *Coarse-graining of condensed phase and biomolecular systems*. CRC Press/Taylor and Francis Group, Boca Raton, 2009. 7, 21

[52] A.Y. Shih, A. Arkhipov, P.L. Freddolino, and K. Schulten. Coarse grained protein-lipid model with application to lipoprotein particles. *The Journal of Physical Chemistry B*, 110(8):3674–3684, 2006. 7, 21

[53] S. Mostaghim, M. Hoffmann, P.H. Konig, T. Frauenheim, and J. Teich. Molecular force field parametrization using multi-objective evolutionary algorithms. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 1, pages 212–219. IEEE, 2004. 7, 21

[54] K. Voltz, J. Trylska, V. Tozzini, V. Kurkal-Siebert, J. Langowski, and J. Smith. Coarse-grained force field for the nucleosome from self-consistent multiscaling. *Journal of Computational Chemistry*, 29(9):1429–1439, 2008. 7, 21

[55] A. Liwo, Y. He, and H.A. Scheraga. Coarse-grained force field: general folding theory. *Physical Chemistry Chemical Physics*, 13(38):16890–16901, 2011. 7, 21

[56] P.J. Hoogerbrugge and J. Koelman. Simulating microscopic hydrodynamic phenomena with dissipative particle dynamics. *EPL (Europhysics Letters)*, 19(3):155, 1992. 7, 21

[57] P. Espanol and P. Warren. Statistical mechanics of dissipative particle dynamics. *EPL (Europhysics Letters)*, 30:191–196, 1995. 7, 21

[58] M. Praprotnik, L.D. Site, and K. Kremer. Multiscale simulation of soft matter: From scale bridging to adaptive resolution. *Annual Review of Physical Chemistry*, 59:545–571, 2008. 7, 21

[59] S.O. Nielsen, R.E. Bulo, P.B. Moore, and B. Ensing. Recent progress in adaptive multiscale molecular dynamics simulations of soft matter. *Physical Chemistry Chemical Physics*, 12(39):12401–12414, 2010. 7, 21

[60] J.H. Park and A. Heyden. Solving the equations of motion for mixed atomistic and coarse-grained systems. *Molecular Simulation*, 35(10):962–973, 2009. 7, 21

[61] M. Praprotnik, S. Matysiak, L.D. Site, K. Kremer, and C. Clementi. Adaptive resolution simulation of liquid water. *Journal of Physics: Condensed Matter*, 19(29):292201, 2007. 7, 21

[62] S.O. Nielsen, P.B. Moore, and B. Ensing. Adaptive multiscale molecular dynamics of macromolecular fluids. *Physical review letters*, 105(23):237802, 2010. 7, 21

[63] B. Ensing, S.O. Nielsen, P.B. Moore, M.L. Klein, and M. Parrinello. Energy conservation in adaptive hybrid atomistic/coarse-grain molecular dynamics. *Journal of Chemical Theory and Computation*, 3(3):1100–1105, 2007. 7, 21

[64] M. Christen and W.F. van Gunsteren. Multigraining: an algorithm for simultaneous fine-grained and coarse-grained simulation of molecular systems. *The Journal of Chemical Physics*, 124:154106, 2006. 8, 21

[65] J.H. Ferziger and M. Perić. *Computational methods for fluid dynamics*. Springer Berlin etc, 1999. 8, 21

[66] J.Q. Broughton, F.F. Abraham, N. Bernstein, and E. Kaxiras. Concurrent coupling of length scales: Methodology and application. *Physical Review B*, 60(4):2391, 1999. 8, 21

[67] A. Nakano, M.E. Bachlechner, R.K. Kalia, E. Lidorikis, P. Vashishta, G.Z. Voyiadjis, T.J. Campbell, S. Ogata, and F. Shimojo. Multiscale simulation of nanosystems. *Computing in Science & Engineering*, 3(4):56–66, 2001. 8, 21

[68] S. Ogata, E. Lidorikis, F. Shimojo, A. Nakano, P. Vashishta, and R.K. Kalia. Hybrid finite-element/molecular-dynamics/electronic-density-functional approach to materials simulations on parallel computers. *Computer Physics Communications*, 138(2):143–154, 2001. 8, 21

[69] D.D. Marsh. *Molecular dynamics-Lattice Boltzmann hybrid method on graphics processors*. PhD thesis, 2010. 8, 21

[70] J.B. Bell, A.L. Garcia, and S.A. Williams. Computational fluctuating fluid dynamics. *ESAIM: Mathematical Modelling and Numerical Analysis*, 44(05):1085–1105, 2010. 8, 21

[71] G. Giupponi, G. De Fabritiis, and P.V. Coveney. Hybrid method coupling fluctuating hydrodynamics and molecular dynamics for the simulation of macromolecules. *The Journal of Chemical Physics*, 126:154903, 2007. 8, 21

[72] O.M. Becker, A.D. MacKerell Jr, B. Roux, and M. Watanabe. *Computational biochemistry and biophysics*. Marcel-Dekker, Inc., New York, 2001. 8, 21

[73] M.B. Ulmschneider, J.P. Ulmschneider, M.S.P. Sansom, and A. Di Nola. A generalized Born implicit-membrane representation compared to experimental insertion free energies. *Biophysical journal*, 92(7):2338–2349, 2007. 8, 21

[74] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987. 8, 22, 31, 89, 130, 151, 155

[75] T. Darden, D. York, and L. Pedersen. Particle mesh Ewald: An Nlog(N) method for Ewald sums in large systems. *Journal of Chemical Physics*, 98:10089–10089, 1993. 8, 22, 89, 130, 151, 155

[76] S. Grudinin and S. Redon. Practical modeling of molecular systems with symmetries. *Journal of Computational Chemistry*, 31(9):1799–1814, 2010. 8, 22, 47, 49, 59, 86, 106, 107, 108, 140

[77] M. Tuckerman, B.J. Berne, and G.J. Martyna. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics*, 97(3):1990, 1992. 8, 10, 22, 23, 130

[78] S. Redon and M.C. Lin. An efficient, error-bounded approximation algorithm for simulating quasi-statics of complex linkages. *Computer-Aided Design*, 38(4):300–314, 2006. 8, 13, 22, 27, 82, 83, 84, 86

[79] S. Redon, N. Galoppo, and M.C. Lin. Adaptive dynamics of articulated bodies. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 936–945. ACM, 2005. 8, 13, 22, 27, 70, 82, 86

[80] M. Bosson, S. Grudinin, X. Bouju, and S. Redon. Interactive physically-based structural modeling of hydrocarbon systems. *Journal of Computational Physics*, 231:2581–2598, 2012. 8, 13, 22, 27

[81] H. Yamashita, S. Endo, H. Wako, and A. Kidera. Sampling efficiency of molecular dynamics and Monte Carlo method in protein simulation. *Chemical Physics Letters*, 342(3):382–386, 2001. 9, 22

[82] W.L. Jorgensen and J. Tirado-Rives. Monte Carlo vs molecular dynamics for conformational sampling. *The Journal of Physical Chemistry*, 100(34):14508–14513, 1996. 9, 22

[83] J.P. Ulmschneider, M.B. Ulmschneider, and A. Di Nola. Monte Carlo vs molecular dynamics for all-atom polypeptide folding simulations. *The Journal of Physical Chemistry B*, 110(33):16733–16742, 2006. 9, 22

[84] B.M. Forrest and U.W. Suter. Generalized coordinate hybrid Monte Carlo. *Molecular Physics*, 82(2):393–410, 1994. 9, 10, 22, 24

[85] Y. Sugita and Y. Okamoto. Replica-exchange molecular dynamics method for protein folding. *Chemical Physics Letters*, 314(1):141–151, 1999. 9, 10, 11, 22, 24, 25, 89, 130, 151, 155

[86] C. Giardina, J. Kurchan, V. Lecomte, and J. Tailleur. Simulating rare events in dynamical processes. *Journal of Statistical Physics*, 145(4):787–811, 2011. 9, 22

[87] E. Hairer, C. Lubich, and G. Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31 of *Springer Series in Computational Mathematics*. Springer Verlag, Berlin, second edition, 2006. 9, 10, 23, 93, 94

[88] J.M. Sanz-Serna. Symplectic integrators for Hamiltonian problems: an overview. *Acta Numerica*, 1(243-286):123–124, 1992. 9, 23

[89] J.A. Izaguirre, S. Reich, and R.D. Skeel. Longer time steps for molecular dynamics. *The Journal of Chemical Physics*, 110:9853, 1999. 10, 23

[90] C.H. Bennett. Mass tensor molecular dynamics. *Journal of Computational Physics*, 19(3):267–279, 1975. 10, 23

[91] K.A. Feenstra, B. Hess, and H.J.C. Berendsen. Improving efficiency of large time-scale molecular dynamics simulations of hydrogen-rich systems. *Journal of Computational Chemistry*, 20(8):786–798, 1999. 10, 23

[92] M. Preto and S. Tremaine. A class of symplectic integrators with adaptive time step for separable Hamiltonian systems. *The Astronomical Journal*, 118(5):2532, 1999. 10, 23

[93] F. Rao and M. Spichty. Thermodynamics and kinetics of large-time-step molecular dynamics. *Journal of Computational Chemistry*, 33(5):475–483, 2012. 10, 23, 128

[94] P. Plechac and M. Rousset. Implicit mass-matrix penalization of hamiltonian dynamics with application to exact sampling of stiff systems. *Multiscale Modeling & Simulation*, 8(2):498–539, 2010. 10, 23

[95] A. Gunaratne. *A penalty function method for constrained molecular dynamics*. PhD thesis, 2006. 10, 24

[96] S. Reich. Modified potential energy functions for constrained molecular dynamics. *Numerical Algorithms*, 19(1):213–221, 1998. 10, 24

[97] A.F. Voter. Hyperdynamics: Accelerated molecular dynamics of infrequent events. *Physical Review Letters*, 78(20):3908–3911, 1997. 10, 24, 89, 130, 151, 155

[98] A. Roitberg and R. Elber. Modeling side chains in peptides and proteins: Application of the locally enhanced sampling and the simulated annealing methods to find minimum

energy conformations. *Journal of Chemical Physics*, 95:9277, 1991. 10, 24, 89, 130, 151, 155

[99] A.Z. Panagiotopoulos. Monte Carlo methods for phase equilibria of fluids. *Journal of Physics: Condensed Matter*, 12(3):R25, 2000. 10, 24

[100] A.H. Widmann and U.W. Suter. Parallelization of a Monte Carlo algorithm for the simulation of polymer melts. *Computer Physics Communications*, 92(2-3):229–251, 1995. 10, 24

[101] E. Marinari and G. Parisi. Simulated tempering: a new Monte Carlo scheme. *EPL (Europhysics Letters)*, 19(6):451, 1992. 10, 24

[102] N.C. Karayiannis, V.G. Mavrantzas, and D.N. Theodorou. A novel Monte Carlo scheme for the rapid equilibration of atomistic model polymer systems of precisely defined molecular architecture. *Physical review letters*, 88(10):105503, 2002. 10, 24

[103] A.F. Voter. Introduction to the kinetic Monte Carlo method. *Radiation Effects in Solids*, 235:1–23, 2007. 10, 24

[104] C.R. Sweet, S.S. Hampton, R.D. Skeel, and J.A. Izaguirre. A separable shadow Hamiltonian hybrid Monte Carlo method. *The Journal of Chemical Physics*, 131(17):174106, 2009. 10, 24

[105] D. Hamelberg, J. Mongan, and J.A. McCammon. Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules. *The Journal of Chemical Physics*, 120:11919, 2004. 10, 24

[106] S. Izrailev, S. Stepaniants, B. Isralewitz, D. Kosztin, H. Lu, F. Molnar, W. Wriggers, and K. Schulten. Steered molecular dynamics. In *Computational molecular dynamics: challenges, methods, ideas*, volume 4, pages 39–65. Springer-Verlag Berlin, 1998. 11, 24

[107] T. Lelievre, G. Stoltz, and M. Rousset. *Free energy computations: A mathematical perspective*. Imperial College Press, London, 2010. 11, 24, 92, 95

[108] C. Shyu and F.M. Ytreberg. Accurate estimation of solvation free energy using polynomial fitting techniques. *Journal of Computational Chemistry*, 32(1):134–141, 2011. 11, 24

[109] B. Lin, K.Y. Wong, C. Hu, H. Kokubo, and B.M. Pettitt. Fast calculations of electrostatic solvation free energy from reconstructed solvent density using proximal radial distribution functions. *The Journal of Physical Chemistry Letters*, 2(13):1626–1632, 2011. 11, 24

[110] D. Brown, H. Minoux, and B. Maigret. A domain decomposition parallel processing algorithm for molecular dynamics simulations of systems of arbitrary connectivity. *Computer Physics Communications*, 103(2):170–186, 1997. 11, 25

[111] P.B. Callahan and S.R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995. 11, 25

[112] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji. 42 TFlops hierarchical N-body simulations on GPUs with applications in both astrophysics and turbulence. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, page 62. ACM, 2009. 11, 25

[113] J.E. Stone, J.C. Phillips, P.L. Freddolino, D.J. Hardy, L.G. Trabuco, and K. Schulten. Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry*, 28(16):2618–2640, 2007. 11, 25

[114] T. Preis, P. Virnau, W. Paul, and J.J. Schneider. GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. *Journal of Computational Physics*, 228(12):4468–4477, 2009. 11, 25

[115] A. Gara, M.A. Blumrich, D. Chen, G.L.T. Chiu, P. Coteus, M.E. Giampapa, R.A. Haring, P. Heidelberger, D. Hoenicke, G.V. Kopcsay, et al. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Development*, 49(2.3):195–212, 2005. 11, 25

[116] D.E. Shaw, M.M. Deneroff, R.O. Dror, J.S. Kuskin, R.H. Larson, J.K. Salmon, C. Young, B. Batson, K.J. Bowers, J.C. Chao, et al. Anton, a special-purpose machine for molecular dynamics simulation. *Communications of the ACM*, 51(7):91–97, 2008. 11, 25

[117] M. Taiji. MDGRAPE-3 chip: a 165 Gflops application specific LSI for molecular dynamics simulations. In *Hot Chips*, volume 16, pages I98–202, 2004. 11, 25

[118] S.M. Larson, C.D. Snow, M. Shirts, et al. Folding@ Home and Genome@ Home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002. 11, 25

[119] B.R. Brooks, C.L. Brooks III, A.D. Mackerell Jr, L. Nilsson, R.J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, et al. CHARMM: the biomolecular simulation program. *Journal of Computational Chemistry*, 30(10):1545–1614, 2009. 11, 25

[120] D.A. Case, T.A. Darden, T.E. Cheatham, C.L. Simmerling, J. Wang, R.E. Duke, R. Luo, R.C. Walker, W. Zhang, K.M. Merz, et al. AMBER 11. *University of California, San Francisco*, 2010. 11, 25

[121] X. Gonze. A brief introduction to the ABINIT software package. *Zeitschrift für Kristallographie*, 220(5/6/2005):558–562, 2005. 11, 25

[122] K.J. Bowers, E. Chow, H. Xu, R.O. Dror, M.P. Eastwood, B.A. Gregersen, J.L. Klepeis, I. Kolossvary, M.A. Moraes, F.D. Sacerdoti, et al. Scalable algorithms for molecular dynamics simulations on commodity clusters. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 43–43. Ieee, 2006. 11, 25

[123] J.C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R.D. Skeel, L. Kale, and K. Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005. 11, 25

[124] W.R.P. Scott, P.H. Hünenberger, I.G. Tironi, A.E. Mark, S.R. Billeter, J. Fennen, A.E. Torda, T. Huber, P. Krüger, and W.F. Van Gunsteren. The GROMOS biomolecular simulation program package. *The Journal of Physical Chemistry A*, 103(19):3596–3607, 1999. 11, 25

[125] B. Hess, C. Kutzner, D. Van Der Spoel, and E. Lindahl. GROMACS 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, 2008. 11, 25

[126] G. Klimeck, M. McLennan, S.P. Brophy, G.B. Adams, and M.S. Lundstrom. nanohub. org: Advancing education and research in nanotechnology. *Computing in Science & Engineering*, 10(5):17–23, 2008. 12, 25

[127] S. Artemova, S. Grudinin, and S. Redon. A comparison of neighbor search algorithms for large rigid molecules. *Journal of Computational Chemistry*, 32(13):2865–2877, 2011. 12, 26, 150, 154

[128] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic acids research*, 28(1):235–242, 2000. 13, 27, 71, 75

[129] S. Artemova, S. Grudinin, and S. Redon. Fast construction of assembly trees for molecular graphs. *Journal of Computational Chemistry*, 32(8):1589–1598, 2011. 13, 27

[130] S. Artemova and S. Redon. Adaptively Restrained Particle Simulations. *Physical Revew Letters*, 109:190201, 2012. 13, 27

[131] G.R. Smith and M.J.E. Sternberg. Prediction of protein-protein interactions by docking methods. *Current opinion in structural biology*, 12(1):28–35, 2002. 15, 31, 89

[132] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov. Principles of docking: an overview of search algorithms and a guide to scoring functions. *Proteins: Structure, Function, and Bioinformatics*, 47(4):409–443, 2002. 15, 31, 89

[133] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. 19

[134] G. Marsaglia and W.W. Tsang. The ziggurat method for generating random variables. *Journal of Statistical Software*, 5(8):1–7, 2000. 19

[135] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 2003. 19

[136] C.J. Cramer. *Essentials of computational chemistry: theories and models*. John Wiley & Sons Inc, 2004. 20

[137] J. Chen, W. Im, and C.L. Brooks III. Application of torsion angle molecular dynamics for efficient sampling of protein conformations. *Journal of Ccomputational Chemistry*, 26(15):1565–1578, 2005. 20

[138] B. Brutovsky, T. Mülders, and G.R. Kneller. Accelerating molecular dynamics simulations by linear prediction of time series. *The Journal of Chemical Physics*, 118(14):6179–6187, 2003. 22

[139] A.K. Mazur. Quasi-hamiltonian equations of motion for internal coordinate molecular dynamics of polymers. *Arxiv preprint physics/9703019*, 1997. 23

[140] J.B. Anderson. *Quantum Monte Carlo: origins, development, applications*. Oxford University Press, USA, 2007. 24

[141] U.H.E. Hansmann and Y. Okamoto. New Monte Carlo algorithms for protein folding. *Current Opinion in Structural Biology*, 9(2):177–183, 1999. 24

[142] G. Turk. *Interactive collision detection for molecular graphics*. PhD thesis, 1989. 31

[143] J.L. Bentley and J.H. Friedman. Data structures for range searching. *ACM Computing Surveys (CSUR)*, 11(4):397–409, 1979. 31

[144] M. Teschner, S. Kimmerle, B. Heidelberger, G. Zachmann, L. Raghupathi, A. Fuhrmann, M.P. Cani, F. Faure, N. Magnenat-Thalmann, W. Strasser, et al. Collision detection for deformable objects. In *Computer Graphics Forum*, volume 24, pages 61–81. Wiley Online Library, 2005. 31

[145] J. Onderik and R. Durikovic. Efficient neighbor search for particle-based fluids. *Journal of the Applied Mathematics, Statistics and Informatics (JAMSI)*, 4(1):29–43, 2008. 31

[146] B.C. Vemuri, Y. Cao, and L. Chen. Fast collision detection algorithms with applications to particle flow. In *Computer Graphics Forum*, volume 17, pages 121–134. Wiley Online Library, 1998. 31

[147] J. Barnes and P. Hut. A hierarchical O(NlogN) force-calculation algorithm. *Nature*, 324(4):446 – 449, 1986. 31

[148] A.W. Appel. Efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6(1):85–103, 1985. 31

[149] I. Carlbom. An algorithm for geometric set operations using cellular subdivision techniques. In *IEEE Computer Graphics and Applications*, pages 44–55. IEEE, 1987. 31

[150] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57. ACM, 1984. 31

[151] S. Gottschalk. *Collision queries using oriented bounding boxes*. PhD thesis, The University of North Carolina, 2000. 31, 32, 36, 38

[152] W.C. Thibault and B.F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 153–162. ACM, 1987. 31

[153] A. Donev, S. Torquato, and F.H. Stillinger. Neighbor list collision-driven molecular dynamics simulation for nonspherical hard particles. I. Algorithmic details. *Journal of Computational Physics*, 202(2):737–764, 2005. 31

[154] G. Paul. A complexity O(1) priority queue for event driven molecular dynamics simulations. *Journal of Computational Physics*, 221(2):615–625, 2007. 31

[155] V.R. de Angulo, J. Cortés, and T. Siméon. BioCD: An efficient algorithm for self-collision and distance computation between highly articulated molecular models. In *Robotics: Science and Systems*. Citeseer, 2005. 31

[156] L. Guibas, A. Nguyen, D. Russel, and L. Zhang. Collision detection for deforming necklaces. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 33–42. ACM, 2002. 31, 36

[157] I. Lotan, F. Schwarzer, D. Halperin, and J.C. Latombe. Efficient maintenance and self-collision testing for kinematic chains. In *Proceedings of the eighteenth annual symposium on Computational geometry*, pages 43–52. ACM, 2002. 31

[158] I. Lotan, F. Schwarzer, D. Halperin, and J.C. Latombe. Algorithm and data structures for efficient energy maintenance during Monte Carlo simulation of proteins. *Journal of Computational Biology*, 11(5):902–932, 2004. 31

[159] W.R. Taylor and Z. Katsimitsoulia. A soft collision detection algorithm for simple Brownian dynamics. *Computational Biology and Chemistry*, 34(1):1–10, 2010. 31

[160] W.F. Van Gunsteren, H.J.C. Berendsen, F. Colonna, D. Perahia, J.P. Hollenberg, and D. Lellouch. On searching neighbors in computer simulations of macromolecular systems. *Journal of Computational Chemistry*, 5(3):272–279, 1984. 31

[161] E.S. Fomin. Consideration of data load time on modern processors for the Verlet table and linked-cell algorithms. *Journal of Computational Chemistry*, 32(7):1386–1399, 2011. 31

[162] V. Yip and R. Elber. Calculations of a list of neighbors in molecular dynamics simulations. *Journal of Computational Chemistry*, 10(7):921–927, 1989. 31

[163] Z. Yao, J.S. Wang, G.R. Liu, and M. Cheng. Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method. *Computer Physics Communications*, 161(1-2):27–35, 2004. 31

[164] Q.F. Fang, R. Wang, and C.S. Liu. Movable hash algorithm for search of the neighbor atoms in molecular dynamics simulation. *Computational Materials Science*, 24(4):453–456, 2002. 31

[165] R.J. Petrella, I. Andricioaei, B.R. Brooks, and M. Karplus. An improved method for nonbonded list generation: Rapid determination of near-neighbor pairs. *Journal of Computational Chemistry*, 24(2):222–231, 2003. 31

[166] P. Gonnet. 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, 2007. 31, 150, 154

[167] W. Mattson and B.M. Rice. Near-neighbor calculations using a modified cell-linked list method. *Computer Physics Communications*, 119(2-3):135–148, 1999. 31

[168] Z.W. Cui, Y. Sun, and J.M. Qu. The neighbor list algorithm for a parallelepiped box in molecular dynamics simulations. *Chinese Science Bulletin*, 54(9):1463–1469, 2009. 31

[169] B.K. Shoichet, I.D. Kuntz, and D.L. Bodian. Molecular docking using shape descriptors. *Journal of Computational Chemistry*, 13(3):380–397, 1992. 31

[170] K.W. Kaufmann, G.H. Lemmon, S.L. DeLuca, J.H. Sheehan, and J. Meiler. Practically Useful: What the Rosetta Protein Modeling Suite Can Do for You. *Biochemistry*, 49(14):2987–2998, 2010. 31

[171] J.T. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *Visualization and Computer Graphics, IEEE Transactions on*, 4(1):21–36, 2002. 31, 32

[172] S. Gottschalk, M.C. Lin, and D. Manocha. OBBTree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 171–180. ACM, 1996. 31, 32, 35

[173] N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *ACM SIGMOD Record*, 19(2):322–331, 1990. 32, 106

[174] N. Megiddo. Linear-time algorithms for linear programming in R3 and related problems. *SIAM Journal on Computing*, 12:759–776, 1983. 32

[175] P.M. Hubbard. Collision detection for interactive graphics applications. In *IEEE Transactions on Visualization and Computer Graphics*, pages 218–230. Published by the IEEE Computer Society, 1995. 32

[176] P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 311–321. Society for Industrial and Applied Mathematics, 1993. 35, 67

[177] G. Barequet, B. Chazelle, L.J. Guibas, J.S.B. Mitchell, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):387–396, 1996. 35

[178] H. Sundar, R.S. Sampath, and G. Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, 2008. 35

[179] E. Ramirez, H. Navarro, R. Carmona, and J. Dos Ramos. Optimizing Collision Detection based on OBB Trees Generated with Genetic Algorithm. In *IV Iberoamerican Symposium in Computer Graphics - SIACG*, pages 1–7, 2009. 36

[180] K. Fischer, B. Gaertner, and M. Kutz. Fast smallest-enclosing-ball computation in high dimensions. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA)*, pages 630–641. Springer, 2003. 36

[181] S. Quinlan. Efficient distance computation between non-convex objects. In *Proceedings of International Conference on Robotics and Automation*, pages 3324–3329. IEEE, 1994. 36

[182] U. Breymann. *Designing Components with the C++ STL*. Addison-Wesley, 1998. 44, 73

[183] B. Bernard and R. Samudrala. A generalized knowledge-based discriminatory function for biomolecular interactions. *Proteins: Structure, Function, and Bioinformatics*, 76(1):115–128, 2009. 44

[184] J.S. Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys (CSUR)*, 33(2):209–271, 2001. 52

[185] L. Arge, G.S. Brodal, and R. Fagerberg. Cache-oblivious data structures. In *Handbook on Data Structures and Applications*, volume 5. Citeseer, 2004. 52

[186] M. Frigo, C.E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *Proceedings of 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*. Published by the IEEE Computer Society, 1999. 52

[187] S.E. Yoon and D. Manocha. Cache-Efficient Layouts of Bounding Volume Hierarchies. *Computer Graphics Forum (Eurographics)*, 25(3):507–516, 2006. 52

[188] T.J. Kim, B. Moon, D. Kim, and S.E. Yoon. RACBVHs: Random-accessible compressed bounding volume hierarchies. *Visualization and Computer Graphics, IEEE Transactions on*, 16(2):273–286, 2010. 52

[189] P. Terdiman. Memory-optimized bounding-volume hierarchies. *http://www.codercorner.com/Opcode.pdf*, 2001. 52

[190] K. Van Workum and J.F. Douglas. Symmetry, equivalence, and molecular self-assembly. *Physical Review E*, 73(3):31502, 2006. 54

[191] I.G. Tironi, R.M. Brunne, and W.F. van Gunsteren. On the relative merits of flexible versus rigid models for use in computer simulations of molecular liquids. *Chemical Physics Letters*, 250(1):19–24, 1996. 54

[192] R. Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114–121. IEEE, 1971. 61, 62, 69

[193] P. Mateti and N. Deo. On algorithms for enumerating all circuits of a graph. *SIAM Journal on Computing*, 5(1):90–99, 1976. 62

[194] G. García, E. Espinosa, I. Ruiz, and M. Gómez-Nieto. Efficient parallel solution to calculate all cycles in graphs. In *Applied Parallel Computing*, pages 765–766. Springer, 2006. 62

[195] G. MacGillivray and M.L. Yu. Generalized partitions of graphs. *Discrete Applied Mathematics*, 91(1-3):143–153, 1999. 62

[196] A. Levin, D. Paulusma, and G.J. Woeginger. The computational complexity of graph contractions I: Polynomially solvable and NP-complete cases. *Networks*, 51(3):178–189, 2008. 62

[197] M.S. Warren and J.K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing*, pages 12–21. ACM, 1993. 67

[198] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in O(NlogN). In *Interactive Ray Tracing 2006, IEEE Symposium on*, pages 61–69. IEEE, 2006. 67

[199] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmayer. Space-filling curves and their use in the design of geometric data structures. *Theoretical Computer Science*, 181(1):3–15, 1997. 68

[200] P.M. Campbell, K.D. Devine, J.E. Flaherty, L.G. Gervasio, and J.D. Teresco. Dynamic octree load balancing using space-filling curves. *Williams College Department of Computer Science, Technical Report CS-03-01*, 2003. 68

[201] T. Ulrich and M. DeLoura. Game programming gems. *Charles River Media*, 2000. 71

[202] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Linear time bounds for median computations. In *Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 119–124. ACM, 1972. 73

[203] R.M. Mukherjee and K.S. Anderson. Orthogonal complement based divide-and-conquer algorithm for constrained multibody systems. *Nonlinear Dynamics*, 48(1):199–215, 2007. 86

[204] R.W. Hockney and J.W. Eastwood. *Computer simulation using particles*. Institute of Physics publishing, Bristol, 1992. 89

[205] J.J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703, 2005. 89

[206] O. Etzmuss, J. Gross, and W. Strasser. Deriving a particle system from continuum mechanics for the animation of deformable objects. *Visualization and Computer Graphics, IEEE Transactions on*, 9(4):538–550, 2003. 89

[207] B. Eberhardt, A. Weber, and W. Strasser. A fast, flexible, particle-system model for cloth draping. *Computer Graphics and Applications, IEEE*, 16(5):52–59, 1996. 89

[208] V.I. Yamakov and E.H. Glaessgen. Nanoscale fracture: To twin or not to twin. *Nature Materials*, 6(11):795–796, 2007. 89, 151, 155

[209] M. Jaraiz, G.H. Gilmer, J.M. Poate, and T.D. De La Rubia. Atomistic calculations of ion implantation in Si: Point defect and transient enhanced diffusion phenomena. *Applied Physics Letters*, 68(3):409–411, 1996. 89

[210] G. Peilert, J. Konopka, H. Stöcker, W. Greiner, M. Blann, and M.G. Mustafa. Dynamical treatment of Fermi motion in a microscopic description of heavy ion collisions. *Physical Review C*, 46(4):1457, 1992. 93

[211] R.D. Engle, R.D. Skeel, and M. Drees. Monitoring energy drift with shadow hamiltonians. *Journal of Computational Physics*, 206(2):432–452, 2005. 94

[212] S.K. Gray, D.W. Noid, and B.G. Sumpter. Symplectic integrators for large scale molecular dynamics simulations: A comparison of several explicit methods. *Journal of Chemical Physics*, 101(5):4062–4072, 1994. 94

[213] I.P. Omelyan, I.M. Mryglod, and R. Folk. Symplectic analytically integrable decomposition algorithms: classification, derivation, and application to molecular dynamics, quantum and celestial mechanics simulations. *Computer Physics Communications*, 151(3):272–314, 2003. 94

[214] M. Tuckerman. *Statistical mechanics: theory and molecular simulation*. Oxford University Press, USA, 2010. 102

[215] A. Rahman. Correlations in the motion of atoms in liquid argon. *Physical Review*, 136(2A):405–411, 1964. 112

[216] B. Adams, M. Pauly, R. Keiser, and L.J. Guibas. Adaptively sampled particle fluids. *ACM Transactions on Graphics (TOG)*, 26(3):48, 2007. 134

[217] F. Bertails, T.Y. Kim, M.P. Cani, and U. Neumann. Adaptive Wisp Tree: a multiresolution control structure for simulating dynamic clustering in hair motion. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 207–213. Eurographics Association, 2003. 136

[218] V.R. de Angulo, J. Cortés, and J.M. Porta. Rigid-CLL: Avoiding constant-distance computations in cell linked-lists algorithms. *Journal of Computational Chemistry*, 33(3):294, 2012. 150, 154