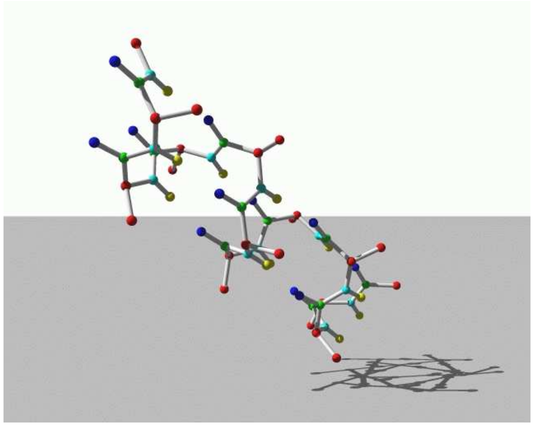
Framework Design, Parallelization and Force Computation in Molecular Dynamics

Thierry Matthey
Department of Informatics
University of Bergen

September 2002



Thesis submitted in partial fulfillment of the requirements for the degree Doctor Scientiarum

Contents

List of Figures

List of Tables

List of Algorithms

List of Programs

1 Introduction

The modeling of phenomena that occur in nature is most frequently formulated by a "fluid-flow" approach. In general particle systems with zero net charge for example, the conservation of energy, mass and momentum together with a force relation typically leads to a set of five coupled differential equations. In principle, these equations can be solved accurately if appropriate boundary conditions are known. Another condition is that the number of real particles inside any computational volume must always be large enough such that macroscopic characteristics (e.g., pressure, density, and temperature) apply. At smaller length scales, or for systems described by discrete building blocks, the particle-like nature of matter has to be taken into account, and one faces a numerical problem of calculating the behavior of N particles – possibly interacting with fluid phases or heat baths. Most frequently, the actual particles of such a system are atoms or molecules. Such systems are most accurately described by quantum theory. Fortunately, for massive particles at sufficient high temperature, quantal effects are negligible small and classical mechanics can be used[1] [84].

Molecular dynamics (MD) [2, 75, 97] describing a *classical* particle molecular system as a function of time has been used for several decades. MD has been successfully applied to understand and explain macro phenomena from micro structures, since it is in many respects similar to real experiments. For example, transport and equilibrium properties of the system in question can be predicted. An MD system is defined by particles (*position* and *momenta*) and their interactions (*potential*), and the *dynamics* of a system is contained in solving Newton's equation of motion, i.e., the classical $N-$body problem.

The classical $N-$body problem lacks a general analytical solution, thus numerical solutions are needed. Solving the dynamics numerically and evaluating the interactions tends to be computationally expensive already for a few thousand particles. Especially, the interactions are generally the computationally dominant part. For large scale MD simulations, we do therefore not only require a powerful machine, but also new algorithmic techniques and parallelization schemes to solve the problem in reasonable time. With the introduction of novel computational algorithms (e.g., multiple time stepping integration schemes, fast electrostatic force algorithms, etc.) and large scale parallel computers, it became possible to study larger systems beyond 10^9 particles [9, 80, 99]. Thus, with such problem

[1]For molecules, the de Broglie wavelength is already very small for very low temperatures; e.g., for a Ca_{40}^{+} Coulomb crystal at 1μK the wavelength is of order 10^{-7}m, a factor 50-100 smaller than the typical separation distance.

sizes certain macroscopic properties of matter can be studied.

There is a proliferation of programs for MD [4, 14, 30, 31, 32, 53, 77, 81, 86, 91, 98, 108, 110, 119, 124, 125] and several of these are robust production codes; some with scalable parallel implementations. They cover common MD problems and are excellent tools to perform simulations. However, many of the codes are legacy programs that are either poorly organized or extremely complex. One important factor is usually the large number of people that contributed to the writing of the codes and the lack of a strong coordination to enforce design, code organization, and programming guide-lines. Most MD applications also suffer from missing documentation that is needed to understand both the design and implementation.

Furthermore, some codes have a long history and were modified multiple times to solve different types of problems at different points in time. Such programs may impose some severe design constraints on future development. They may also use different programming languages and system dependent libraries. Moreover, in the past, the design, the readability and also the portability has been sacrificed to a certain level, due to aggressive manual optimization techniques and tricks to obtain high performance. Nowadays compilers are able to perform such optimizations to a certain extent.

Finally, a parallel implementation adds an extra complexity level, since these MD programs usually force parallelism throughout the whole application. As an algorithm development platform they are inappropriate to test novel algorithms. The lack of a suitable development platform has not only a negative effect on students and junior researchers, but also on non-core computer science researchers, who have to spend considerable time dealing with these difficulties. At the very end, it does also affect the rate of dissemination of new results; for example, many new algorithms for MD appear in leading journals each month, but their implementations in MD codes appear only several years later, if at all (e.g., [8]).

In the past, the main focus for MD programs has been on producing well-performing and robust production codes. As discussed above, codes of such applications often have a high threshold with respect to *testing* and *comparing new algorithms* and lack general, built-in comparison facilities. However, the development from scratch is very time consuming, and most of the time is spent in implementing supporting code that is only marginally related to the real problem of interest. To overcome this dilemma, a flexible and easy extendable platform for research is needed. This will motivate researchers to reuse functionalities and extend the platform with new algorithms or applications. An example is the software package BALL [12] that has special emphasis on molecular modeling and its representation by data structures.

There are different approaches that enable reuse of code, and according to Coulange [25] and Meyer [85], the object-oriented approach is the most favorable one. Thus, to satisfy the requirement of flexibility and code reuse, an object-oriented *component-based framework*[2,3] is proposed. The purpose of the component-based framework is to support both white- and black-box design. White-box design is based on inheritance to adapt to a particular problem. Design and implementation have to be well known, but provide great flexibility. In a black-box design, components are reused by composition, which does not demand too much knowledge of the framework, but it comes with less flexibility. Run-time efficiency, however, is a crucial issue for MD programs. The framework must address this from the beginning of the design and be aware of language specific pitfalls [18, 115]. Furthermore, the framework must be designed to support transparent parallelization, such that developers are not forced to deal with parallelism issues, even in a parallel environment.

The first part of the thesis is concerned with the optimization and extension of the particular MD prototype SPRINGS developed by Jan Petter Hansen[4]. This work was carried out to get some valuable insight information about MD simulations and requirements from MD application users. SPRINGS is also used to investigate parallelization schemes for MD applications, especially the usage of new features of the MPI-2 standard, e.g., the one-sided communication mechanism.

The second and main part of the thesis consists of the design and development of a multi-purpose framework for MD applications, with emphasis on design and implementation of force algorithms. The requirements to a novel MD framework were motivated by the knowledge of MD obtained in the first part and the evaluation of advantages and drawbacks of existing MD programs. The main goal was to develop a flexible, extendable MD research platform, neither excluding run-time efficiency nor parallelism. The design of the framework and its implementation PROTOMOL[5] [63] were realized in collaboration with Jesús A. Izaguirre[6] and his students. Furthermore, some design aspects were introduced from previous work with the object- and component-oriented graphics framework

[2] "A set of classes that embodies an abstract design for solutions to a family of related problems." [65]

[3] "Good frameworks can be used for things that the designer never dreamed of." [64]

[4] Department of Physics, University of Bergen.

[5] **PROTO**type **MOL**ecular dynamics

[6] Department of Computer Science and Engineering, University of Notre Dame, USA.

BOOGA[7] [3, 17, 83, 112].

1.1 Structure of the thesis

This dissertation describes the component-based framework PROTOMOL and details the most essential force algorithms and parallelization aspects. Section 2 starts with an introduction of the physical and numerical background of MD systems. In Section 3, an overview of force computation is presented, with emphasis on fast electrostatic force algorithms. Then, in Section 4, different parallelization approaches with focus on MD simulations are discussed; some aspects are discussed in more detail in the appended publications [C] and [D]. Section 5 details the design and implementation of the component-based framework PROTOMOL. Furthermore, the section gives some practical examples of the extendibility and optimization that is present in PROTOMOL. Section 6 contains conclusions and possible directions for future work.

Appendix A is the collection of papers published during this thesis. The publications [A] and [B] are MD studies of large scale 2-dimensional physical problems. The simulations were performed with the prototype MD program SPRINGS. Paper [C] is a detailed study of MPI's one-sided communication mechanism, a feature of the MPI-2 standard. Publication [D] explains an incremental parallelization approach and its implementation into PROTOMOL.

Paper A: T. Matthey and J. P. Hansen. Molecular dynamics simulations of sliding friction in a dense granular material. *Modelling and Simulation in Materials Science and Engineering*, 6(6):701–707, November 1998.

Paper B: I. Skauvik, J. P. Hansen, and T. Matthey. Dynamics of clustering in a binary Lennard-Jones material. *Modelling and Simulation in Materials Science and Engineering*, 8(5):665–676, September 2000.

Paper C: T. Matthey and J. P. Hansen. Evaluation of MPI's one-sided communication mechanism for short-range molecular dynamics on the Origin2000. In T. Sørevik, R. Moe, A. H. Gebremedhin and F. Manne, editors, *Applied Parallel Computing. New Paradigms for HPC in Industry and Academia*, volume 1947 of *In Lecture Notes in Computer Science*, pages 374–365, Bergen, Norway, June 2000. Springer-Verlag.

[7]**B**erne's **O**bject–**O**riented **G**raphics **A**rchitecture

Paper D: T. Matthey and J. A. Izaguirre. ProtoMol: A molecular dynamics framework with incremental parallelization. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing (PP01)*, Proceedings in Applied Mathematics, Portsmouth, Virginia, March 2001. Society for Industrial and Applied Mathematics (SIAM).

Appendix B contains detailed diagrams of the design and the hierarchy of the forces. Appendix C includes the implementation code of an actual parallelized force. Appendix D.1 contains snapshots of four different molecular systems. Appendix D.2 gives an example of PROTOMOL used in a student course. Appendices D.3 and D.4 round up the thesis with a gallery of results from ongoing project collaborations on applications of PROTOMOL to very different sets of particle systems.

2 Physical and Numerical Model

In classical MD simulations the dynamics are described by Newton's equation of motion

$$m_i \frac{\partial^2}{\partial t^2} \vec{x}_i(t) = \vec{F}_i(t), \tag{1}$$

where m_i is the mass of atom[8] i, $\vec{x}_i(t)$ the atomic position at time t and $\vec{F}_i(t)$ the instant force on the atom. Note that there are other formulations to describe the dynamics of an MD system, e.g., Lagrangian definition of classical mechanics [97, pp. 42-44, Ch. 6] or Hamilton's equation [47]. Where Newton's equation of motion describes nature (to a certain extent) conserving the energy (NVE), these advanced formulations allow to modify the dynamics to achieve equilibrium states satisfying certain specified requirements, e.g., constant temperature (NVT), constant pressure (NVP) or rigid molecules. These approaches are more for computational convenience and beyond the scope of this thesis; they were partly implemented into PROTOMOL [60] by other developers.

The force $\vec{F}_i$ is defined by the potential energy

$$\vec{F}_i = -\nabla_i U(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N) + \vec{F}_i^{\text{extended}}, \tag{2}$$

where U is the potential, $\vec{F}_i^{\text{extended}}$ an extended force (e.g., velocity-based friction) and N the total number of atoms in the system. Typically, the potential is given by

$$\begin{aligned} U &= U^{\text{bonded}} + U^{\text{non-bonded}} + U^{\text{external}} && (3) \\ U^{\text{bonded}} &= U^{\text{bond}} + U^{\text{angle}} + U^{\text{dihedral}} + U^{\text{improper}} && (4) \\ U^{\text{non-bonded}} &= U^{\text{electrostatic}} + U^{\text{Lennard-Jones}}. && (5) \end{aligned}$$

The bonded forces are a sum of $\mathcal{O}(N)$ terms. The non-bonded forces are a sum of $\mathcal{O}(N^2)$ terms due to the pair-wise definition. U^{external} covers additional interactions, e.g., electromagnetic fields ($\mathcal{O}(N)$) or even gravitation ($\mathcal{O}(N^2)$). U^{bond}, U^{angle}, U^{dihedral} and U^{improper} define the covalent bond interactions to model flexible molecules. In this thesis, the bonded force terms are based on harmonic potentials and mimic physical effects based on quantum mechanics. Note that there are other formulations to model these terms, which are not harmonic necessarily, e.g.,

[8] Atom is here used as an indivisible entity and interchangeably used with any massive particle.

Morse-potentials. $U^{\text{electrostatic}}$ represents the well-known Coulomb potential and $U^{\text{Lennard-Jones}}$ models a van der Waals attraction and a hard-core repulsion. The coefficients for $U^{\text{Lennard-Jones}}$ can be determined experimentally and/or theoretically, e.g., by *ab initio* quantum mechanical calculations.

The potential for bond interactions is described as a linear bond between two atoms by a simple harmonic spring

$$U_m^{\text{bond}} = \frac{1}{2} K_B \left(||\vec{x}_{ij}|| - l_m \right)^2 , \tag{6}$$

where K_B is a bond force constant, $\vec{x}_{ij} = \vec{x}_j - \vec{x}_i$ is the distance vector between atoms i and j, and l_m is the equilibrium bond length between the atoms i and j; m denotes the bond index. The potential for angle interactions is described as an angular bond between three atoms by a harmonic angular spring

$$U_m^{\text{angle}} = \frac{1}{2} K_A \left(\theta_{ijk} - \theta_m \right)^2 , \tag{7}$$

where K_A is an angle force constant, θ_{ijk} the current angle, and θ_m the reference angle for constraint m. Some systems also include the so-called Urey-Bradley term, a harmonic spring between atoms i and k; $K_{\text{UB}} \left(||\vec{x}_{ik}|| - r_{\text{UB}} \right)^2$, where K_{UB} and r_{UB} are Urey-Bradley parameters. Dihedral and improper (also known as torsion) bonded forces [75] define the interaction (torsion) between four bonded atoms. They are modeled by an angular spring between planes formed by the first three atoms and the second three atoms. The potential for a dihedral or improper force between atoms i, j, k, and l is given by

$$U_m^{\text{dihedral}}, U_m^{\text{improper}} = \begin{cases} K_t \left(1 + \cos(n\phi + \delta) \right) & \text{if } n > 0 \\ K_t \left(\phi - \delta \right)^2 & \text{if } n = 0 \end{cases} , \tag{8}$$

where K_t is a torsion force constant, ϕ is the angle between the planes formed by the atoms i, j, k and j, k, l, respectively. n is the periodicity and δ the phase shift of the bond. The potential function for an electrostatic pair-wise interaction is given by

$$U_{ij}^{\text{electrostatic}} = \frac{1}{4\pi\epsilon_0} \frac{q_i q_j}{||\vec{x}_{ij}||}, \tag{9}$$

where ϵ_0 is the permittivity of free space constant, and q_i and q_j are the charges of atoms i and j, respectively. Figure 1 shows the slowly decaying electrostatic interaction with normalized constants. The potential for the Lennard-Jones[9] (LJ) pair-wise interaction is

$$U_{ij}^{\text{Lennard-Jones}} = \frac{A_{ij}}{||\vec{x}_{ij}||^{12}} - \frac{B_{ij}}{||\vec{x}_{ij}||^{6}}, \tag{10}$$

where $A_{ij} \geq 0$ and $B_{ij} \geq 0$ are the LJ parameters for atoms i and j. They define the energy minimum and the cross-over point, where the LJ function is zero. Figure 2 shows the attractive and repulsive part of the LJ function for A, $B = 1$.

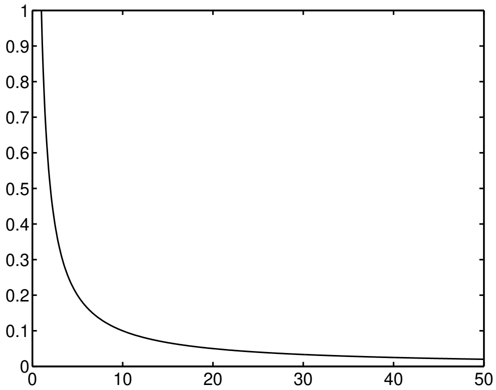


Figure 1: Coulomb: $y = x^{-1}$.

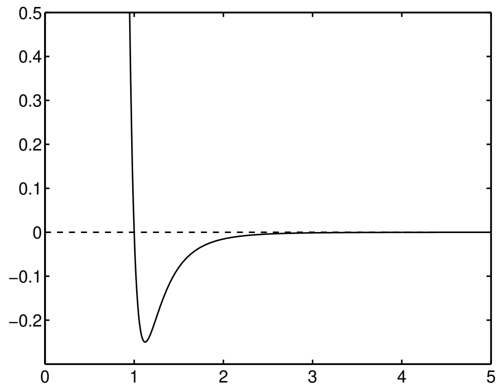


Figure 2: LJ: $y = x^{-12} - x^{-6}$.

2.1 Numerical integration

Newton's equation of motion is an ordinary differential equation of second order. Its integration is often solved by the numerical Leap-Frog[10] method, which is symplectic[11] [127] and time reversible. Despite its low order, it has excellent energy conservation properties and is computationally cheap. With Δt as time step, x^t and v^t the coordinate and the velocity at time t, respectively, a single integration step is given by

$$v^{t+\frac{1}{2}} = v^{t-\frac{1}{2}} + \frac{\Delta t}{m} F^t \tag{11}$$

$$x^{t+1} = x^t + \Delta t v^{t+\frac{1}{2}}. \tag{12}$$

[9] Also called London-van der Waals interaction.
[10] The Leap-Frog and the velocity Verlet method are algebraically equivalent.
[11] The Jacobian of the integrator is symplectic.

The local error of coordinates and velocities is of order $\mathcal{O}(\Delta t^4)$ and $\mathcal{O}(\Delta t^2)$, respectively. The name 'Leap-Frog' has its origin from the fact that the coordinates and the velocities are interleaved in time. If necessary, v^{t+1} can be obtained by

$$v^{t+1} = v^{t+\frac{1}{2}} + \frac{\Delta t}{2m} F^{t+1}. \tag{13}$$

By reformulating the Leap-Frog and adding the force evaluation, we can describe one complete time step ($(x^t, v^t) \rightarrow (x^{t+1}, v^{t+1})$) of an MD simulation as

$$v^{t+\frac{1}{2}} = v^t + \frac{\Delta t}{2m} F^t \qquad \text{(half a kick)} \tag{14}$$

$$x^{t+1} = x^t + \Delta t v^{t+\frac{1}{2}} \qquad \text{(a drift)} \tag{15}$$

$$F^{t+1} = -\nabla U(x^{t+1}) + F^{t+1}_{\text{extended}} \qquad \text{(evaluate forces)} \tag{16}$$

$$v^{t+1} = v^{t+\frac{1}{2}} + \frac{\Delta t}{2m} F^{t+1} \qquad \text{(half a kick).} \tag{17}$$

A complete MD simulation is described by Algorithm 1. It consists basically of the loop solving the equation of motion numerically and the evaluation of forces on each particle, and some additional pre- and post-processing.

MD Simulation:

1. Construct initial configuration of x^0, v^0 and F^0;
2. **loop** 1 **to** number of steps
   (a) Update velocities – Eq. (14);
   (b) **Evaluate** forces on each particle – Eq. (15); (see Algorithm 3)
   (c) Update positions – Eq. (16);
   (d) Update velocities – Eq. (17);
3. Post-processing

Algorithm 1: Pseudo-code of an MD simulation.

When integrating Newton's equation of motion numerically, the fastest motions restrict the time step Δt. For the Leap-Frog method, this is typically 1 femtosecond (1 fs = 10^{-15} s), assuming a typically MD system of biomolecules[12].

[12]Molecules of biological origin or relevance.

MD systems consist in general of different force types, which also have different time scales of dynamics. Velocities of molecules or single atoms are typically a factor of 10-50 slower than the intra-molecular motions. Multiple time stepping (MTS) integrators address the different time scales by splitting the forces – if possible – by frequencies and integrating them with appropriate time steps. The splitting of forces is defined by switching functions, discussed in the next section. MTS has been used to lengthen the time step for most of the interactions in the equation of motion. Furthermore, one also hopes that the low frequency force part is the computationally most expensive part, since it is evaluated less frequently. MTS is therefore a remedy to achieve better performance, but it comes with some additional computational work.

A typical MTS integrator is the Verlet-I/r-RESPA [49, 118] method, described in Algorithm 2. The method consists of splitting the force contributions on a tagged particle into fast and slow components. If these contributions are completely decoupled, it is possible to integrate the equations of motion with different time steps which are appropriate for the slow and fast dynamics. Slow components are taken into account only every N time steps, where they act as a kind of impulse, acting on the particle. A detailed evaluation and description of integrators and MTS methods are beyond the scope of this thesis; the reader is referred to [7, 39, 60, 57, 79, 129].

In particular, researchers at the University of Notre Dame have implemented

$v^t = v^t + \frac{\Delta t N}{2m} F^t_{\text{slow}}$ (half a kick)

for $k = 0, \ldots, N-1$ do

$\quad v^{t+\frac{k+\frac{1}{2}}{N}} = v^{t+\frac{k}{N}} + \frac{\Delta t}{2m} F_{\text{fast}}^{t+\frac{k}{N}}$ (half a kick)

$\quad x^{t+\frac{k+1}{N}} = x^{t+\frac{k}{N}} + \Delta t v^{t+\frac{k+\frac{1}{2}}{N}}$ (a drift)

$\quad F_{\text{fast}}^{t+\frac{k+1}{N}} = -\nabla U_{\text{fast}}(x^{t+\frac{k+1}{N}})$ (evaluate fast forces)

$\quad v^{t+\frac{k+1}{N}} = v^{t+\frac{k+\frac{1}{2}}{N}} + \frac{\Delta t}{2m} F_{\text{fast}}^{t+\frac{k+1}{N}}$ (half a kick)

end do

$F^{t+1}_{\text{slow}} = -\nabla U_{\text{slow}}(x^{t+1})$ (evaluate slow forces)

$v^{t+1} = v^{t+1} + \frac{\Delta t N}{2m} F^{t+1}{}_{\text{slow}}$ (half a kick).

Algorithm 2: The Verlet-I/r-RESPA method.

MTS integrator algorithms, that coupled with the fast electrostatic solvers and an incremental parallelization approach (both implemented for this thesis), represent a substantial speedup.

2.2 Switching functions

Switching functions ($SW(r)$) are primarily for computational convenience. These functions modify (or filter) an original potential and its force contributions to achieve certain properties (Figure 3). They are typically used for the pair-wise non-bonded interactions and are distance based. The modification of the potential and force contributions is given by

$$U'_{ij} = SW(||\vec{x}_{ij}||)U_{ij} \tag{18}$$

$$F'_{ij} = F_{ij}SW(||\vec{x}_{ij}||) - U'_{ij}\nabla SW(||\vec{x}_{ij}||) \tag{19}$$

The most important switching functions are:

- Truncation (or shift): The potential is truncated at a certain cutoff distance and is zero beyond. A shift removes the C^0-discontinuity at cutoff distance, but the force is still discontinuous (see Shifted in Figure 3).
- Modification: This brings the potential and forces smoothly to zero at the cutoff distance to avoid any discontinuities (see Modified in Figure 3).
- Range: It filters the potential and forces for a given pair distance range.

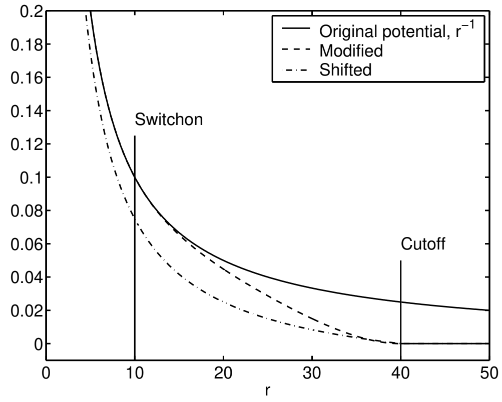


Figure 3: Two switching functions modifying the original potential r^{-1}.

Switching functions are often used in conjunction with MTS to split forces into slow and fast varying parts.

2.3 Boundary conditions

For a complete definition of the MD model, one needs to specify boundary conditions to define how atoms or molecules interact with their surrounding. The selection of boundary conditions depends on the physical system and the questions of matter. The most relevant boundary conditions are:

- Vacuum: no boundary conditions at all. It is generally used to test algorithms or for studies of single molecule systems.
- Periodic boundary conditions: particles leaving the unit MD cell are wrapped around, re-entering the unit MD cell on the opposite side. It is used to simulate an infinite system with a small-sized system, e.g., bulk liquids.
- Spherical or cylindrical boundary conditions (or constraints): convenient for systems which fit the geometrical structure and to preserve the spatial structure, e.g., water droplets. Typically, they are implemented by external forces.

A full discussion of boundary conditions can be found in [2, pp. 24-32].

3 Force Computation

The force evaluation in Algorithm 1 is defined by the sum of interactions in Eq. (3), where the particles interact with each other. By expanding point 2b in Algorithm 1 recursively, the force evaluation for a typical MD simulation can be described as in Algorithm 3. Note that for some particular MD simulation some forces may not apply, e.g., SPRINGS considers only systems with Lennard-Jones interactions and an optional velocity-based friction (see publications [A] and [B]).

Evaluation of forces:

- Bond forces – Eq. (6);
- Angle forces – Eq. (7);
- Dihedral forces – Eq. (8);
- Improper forces – Eq. (8);
- Electrostatic forces – Eq. (9);
- Lennard-Jones forces – Eq. (10);
- Optional or other forces;

Algorithm 3: Pseudo-code of the force evaluation.

A force type acting on one particular particle may in some cases only depend on the particle itself (e.g., friction), but other force types are defined as pair-wise interactions or by n-tuples, so-called $n-$body interactions. General MD applications distinguish between bonded and non-bonded forces. The topology of bonded forces are of static type and do in general not change during the whole simulation, so they can be described with simple static graph representations. They typically model the intra-molecular forces, which are short-range interactions. The bonded forces are a sum of $\mathcal{O}(N)$ terms. The non-bonded forces describe interactions between particles of dynamic behavior and are generally defined by a pair-wise potential in Eqs. (9) and (10). These interactions cannot be determined statically. Generally, they are of a long-range nature and dominate the force calculation by $\mathcal{O}(N^2)$ complexity. Thus, they are the most computational critical part and optimizations happen here.

One common optimization method for non-bonded force evaluations is to truncate at a given cutoff distance (r_c) to limit the number of pair-wise interactions, only considering the closest neighbors. This can be achieved using a cell list algorithm [9][97, pp. 47-52] (see also paper [C]), or a list of neighbors [97, pp. 52-55]. However, simple truncation leads to discontinuities in the forces and it violates energy conservation for symplectic integrators. One way to handle cutoffs are switching functions, which smoothly bring the energy and the forces to zero at the cutoff distance to avoid any discontinuities. Truncation together with switching functions performs with $\mathcal{O}(N)$ time complexity and works well for fast decaying potentials, where the long-range part – beyond the cutoff distance – is negligible, e.g., van der Waals interactions (Figure 2). Furthermore, one can account for the neglected energy contributions in van der Waals potentials (or other short-range potentials) by applying a long-range correction, which assumes that the radial distribution function $g(r) = 1; r > r_c$ (cf. [2]).

For long-range interactions, where the contributions decay slowly (Figure 1) and the long-range part may have a significant impact on the simulation results, we need advanced algorithms to avoid the direct method with $\mathcal{O}(N^2)$ time complexity. In MD, the electrostatic interactions are the most difficult problem, whereas van der Waals interactions are normally handled by cutoff techniques. Considering also the fact that the numerical integration itself is of order $\mathcal{O}(N)$, one can conclude that the most critical part in an MD simulation are the electrostatic interactions in Algorithm 3.

3.1 Fast electrostatic force algorithms

For systems with partial charges or higher multipoles, the electrostatic interactions play a dominant role, e.g., in protein folding, ligand binding, and ion crystals. The most relevant algorithms to solve electrostatic interaction in less than $\mathcal{O}(N^2)$ time complexity are discussed in Sections 3.2-3.5. Three of these methods were implemented (see Section 5.7).

3.2 Standard Ewald summation

A general potential energy function U of a system of N particles with an interaction potential $\phi(\vec{x}_{ij} + \vec{n})$ and periodic boundary conditions can be expressed as

$$U = \frac{1}{2} \sum_{\vec{n}}^{\dagger} \sum_{i=1}^{N} \sum_{j=1}^{N} \phi(\vec{x}_{ij} + \vec{n}), \tag{20}$$

where $\sum_{\vec{n}}$ is the sum over all lattice vectors $\vec{n} = (L_x n_x, L_y n_y, L_z n_z)$, $n_{x,y,z} \in \mathbb{N}$ and $L_{x,y,z}$ are the dimensions of the unit MD cell and $\vec{x}_{ij} = \vec{x}_j - \vec{x}_i$. The "daggered" (†) summation indicates the exclusion of all pairs $i = j$ inside the original unit MD cell ($\vec{n} = \vec{0}$). Most molecular force fields provide exclusion schemes to exclude additional pairs, e.g., the so-called *1-3 exclusion* excludes directly connected pairs and pairs with a direct common neighbor. Furthermore, some exclusion schemes may also modify the potential by a factor for certain pairs.

If the potential ϕ satisfies

$$|\phi(\vec{x})| \leq A|\vec{x}|^{-3-\epsilon} \tag{21}$$

for large enough $\vec{x}$ and $A > 0$ and $\epsilon > 0$, then the sum in Eq. (20) is absolute convergent[13].

Inequality (21) is not satisfied by the Coulomb potential $\phi(\vec{x}_{ij}) = cq_iq_j|\vec{x}_{ij}|^{-1}$, such that the infinite lattice sum in Eq. (20) is only conditionally convergent[14]. In case of the Lennard-Jones potential, the sum is absolutely convergent.

The well-known Ewald summation method [34] is in general useful in systems with large, spatial potential differences, where the summation over one unit cell does not converge sufficiently, i.e., the lattice sum is not absolutely convergent. The lattice sum with the Coulomb potential is usually expressed as

$$U^{\text{electrostatic}} = \frac{1}{4\pi\epsilon_0} \frac{1}{2} \sum_{\vec{n}}^{\dagger} \sum_{i=1}^{N} \sum_{j=1}^{N} \frac{q_i q_j}{|\vec{x}_{ij} + \vec{n}|}. \tag{22}$$

To overcome the conditionally and insufficient convergence of Eq. (22), the sum is split into two parts by the following trivial identity

$$\frac{1}{r} = \frac{f(r)}{r} + \frac{1 - f(r)}{r}. \tag{23}$$

The basic idea is to separate the fast variation part for small r and the smooth part for large r. In particular, the first part should decay fast and be negligible

[13]The sum converges and does not depend on the order of summation.

[14]The convergence of the sum depends on the order of summation.

beyond some cutoff distance, whereas the second part should be smooth for all r, such that its Fourier transform can be represented by a few terms. Ewald [34] suggested to choose a Gaussian convergence factor $g(s,\vec{n}) = e^{-s|\vec{n}|^2}$ such that the sum becomes (see [76] for a detailed mathematical derivation of the Ewald summation)

$$\begin{aligned} U^{\text{electrostatic}} \quad = \quad & \underbrace{\frac{1}{4\pi\epsilon_0}\frac{1}{2}\sum_{\vec{n}}{}^{\dagger}\sum_{i=1}^{N}\sum_{j=1}^{N} q_i q_j \frac{\operatorname{erfc}(\alpha|\vec{x}_{ij}+\vec{n}|)}{|\vec{x}_{ij}+\vec{n}|}}_{\text{Real-space term}} \\ + \quad & \underbrace{\frac{1}{\epsilon_0 V}\frac{1}{2}\sum_{\vec{k}\neq 0}\frac{1}{\vec{k}^2}e^{-\frac{\vec{k}^2}{4\alpha^2}}\left[\left|\sum_{i=1}^{N} q_i\cos(\vec{k}\cdot\vec{x}_i)\right|^2 + \left|\sum_{i=1}^{N} q_i\sin(\vec{k}\cdot\vec{x}_i)\right|^2\right]}_{\text{Reciprocal-space term}} \\ - \quad & \underbrace{\frac{1}{4\pi\epsilon_0}\frac{1}{2}\sum_{j=1}^{M}{}^{\dagger^{-1}}\sum_{k=1}^{N_j}\sum_{l=1}^{N_j} q_{j_k} q_{j_l}\frac{\operatorname{erf}(\alpha|\vec{x}_{j_k j_l}|)}{|\vec{x}_{j_k j_l}|}}_{\text{Intra-molecular self energy}} \\ - \quad & \underbrace{\frac{\alpha}{4\pi^{\frac{3}{2}}\epsilon_0}\sum_{i=1}^{N} q_i^2}_{\text{Point self energy}} \quad - \quad \underbrace{\frac{1}{8\epsilon_0 V\alpha^2}\left|\sum_{i=1}^{N} q_i\right|^2}_{\text{Charged system term}} \quad . \end{aligned} \tag{24}$$

Here, α is the splitting parameter of the real and reciprocal part. For an optimal α the Ewald summation scales as $\mathcal{O}(N^{\frac{3}{2}})$ [34, 36, 89] in Eq. (26). $\dagger^{-1}$ is the "inverse daggered" summation. The intra-molecular self term corrects interactions on the same molecule, which are implicitly included in the reciprocal-space term, but are not required in the exclusion model. Self interactions are canceled out by the self point term. The charged system term is only necessary if the total net charge of the system is non-zero. Note, that some MD systems have an additional surface dipole term to model dipolar systems more accurately [76, 122]. This term is not suited for mobile ions, since it will create discontinuities in the energy and force contributions when ions cross boundaries.

The electrostatic force on particle i is given by

$$\begin{aligned}
F_i^{\text{electrostatic}} &= -\nabla_{\vec{x}_i} U^{\text{electrostatic}} \\
&= \underbrace{\frac{q_i}{4\pi\epsilon_0} \sum_{\vec{n}}^{\dagger} \sum_{j=1}^{N} q_j \left[\frac{\operatorname{erfc}(\alpha|\vec{x}_{ij}+\vec{n}|)}{|\vec{x}_{ij}+\vec{n}|} + \frac{2\alpha}{\sqrt{\pi}} e^{-\alpha^2|\vec{x}_{ij}+\vec{n}|^2} \right] \frac{\vec{x}_{ij}+\vec{n}}{|\vec{x}_{ij}+\vec{n}|^2}}_{\text{Real-space term}} \\
&+ \underbrace{\frac{1}{\epsilon_0 V} \sum_{\vec{k}\neq 0} q_i \frac{\vec{k}}{\vec{k}^2} e^{-\frac{\vec{k}^2}{4\alpha^2}} \left[\sin(k \cdot r_i) \sum_{j=1}^{N} q_j \cos(\vec{k}\cdot\vec{x}_j) - \cos(k \cdot r_i) \sum_{j=1}^{N} q_j \sin(\vec{k}\cdot\vec{x}_j) \right]}_{\text{Reciprocal-space term}} \\
&+ \underbrace{\frac{q_i}{4\pi\epsilon_0} \frac{1}{2} \sum_{j}^{\dagger^{-1}} q_j \left[\frac{2\alpha}{\sqrt{\pi}} e^{-\alpha^2|\vec{x}_{ij}|^2} - \frac{\operatorname{erf}(\alpha|\vec{x}_{ij}|)}{|\vec{x}_{ij}|} \right] \frac{\vec{x}_{ij}}{|\vec{x}_{ij}|^2}}_{\text{Intra-molecular term}}
\end{aligned} \tag{25}$$

Furthermore, the Ewald method can also be used for van der Waals interactions [21, 67] and several other potentials [43, pp. 237-256],[76]. The computational efficiency and accuracy for 2-dimensional periodic boundary conditions in a 3-dimensional system is discussed in [69]. The case with 1-dimensional periodic boundary conditions is addressed in [74]. To some extent, vacuum systems can be treated by imposing periodic boundary conditions. The dimensions of the original unit cell are chosen big enough, such that the contributions between cell images are negligible. In practice, the dimensions of the unit cell are a factor of 100 larger than the minimal bounding box of particles.

Choice of parameters

The accuracy and performance of the Ewald summation is governed by the splitting factor α, the real- and reciprocal-space term cutoffs r_c and k_c, and the accuracy ϵ. The splitting parameter α defines how fast the sums converge and defines the cutoffs r_c and k_c for a given accuracy ϵ. For most applications, r_c is small enough such that $\vec{n} \in \{\vec{0}\}$, which also simplifies the sum for the real-space term. Both the real- and reciprocal-space term converge rapidly, such that only a few terms need to be considered. For the real part only terms satisfying $|\vec{x}_{ij}+\vec{n}| < r_c$ are included, whereas for the reciprocal part only summands with $|\vec{k}| < k_c$ are evaluated.

From Eq. (24) it is obvious that for a given accuracy and a fixed α the required work scales as $\mathcal{O}(N^2)$ for the reciprocal term and as $\mathcal{O}(N)$ for the real term. To achieve an overall work complexity of $\mathcal{O}(N^{\frac{3}{2}})$, α must vary with N

$$\alpha = c\sqrt{\pi}\left(\frac{N}{V^2}\right)^{\frac{1}{6}} \tag{26}$$

$$r_c = \frac{\sqrt{-\ln\epsilon}}{\alpha} \tag{27}$$

$$k_c = 2\alpha\sqrt{-\ln\epsilon}\,. \tag{28}$$

Here, V is the volume and the constant c determines the ratio of execution time of the real and reciprocal term, which may vary from one platform to another. The standard Ewald summation is unsurpassed for very high accuracy. It is relatively easy to implement and the desired accuracy can be increased and controlled up to machine precision without any additional programming effort (see Figure 11). Due to these excellent properties, the Ewald method is often used as reference for the evaluation of other methods with periodic boundary conditions. A more detailed discussion for the optimal choice of α and more accurate error estimates can be found in [27, 36, 71, 90].

3.3 Mesh-based Ewald methods

The mesh-based Ewald methods approximate the reciprocal-space term of the standard Ewald summation by a discrete convolution on an interpolating grid, using the discrete Fast-Fourier transforms (FFT). By choosing an appropriate splitting parameter α, the computational cost can be reduced from $\mathcal{O}(N^{\frac{3}{2}})$ to $\mathcal{O}(N\log N)$. The accuracy and speed are additionally governed by the mesh size and the interpolation scheme, which makes the choice of optimal parameters more difficult. At present, there exist several implementations based on this idea, but they differ in detail. In [27] three essential methods are compared and summarized: particle-particle-particle-mesh [54] (P^3M), particle-mesh Ewald [26] (PME) and smooth particle-mesh Ewald [33] (SPME).

Unfortunately, the mesh-based Ewald methods are affected by errors when performing interpolation, FFT, and differentiation [90]. However, it would be misleading to infer that these methods sacrifice accuracy in favor of run-time performance.

For the fast mesh-based Ewald methods, there exists a critical number N^* such that they are faster than the standard Ewald method for $N > N^*$, due to the fact of better scaling. In [70], the computational efficiency and accuracy with 2-dimensional periodic boundary conditions for 3-dimensional systems are discussed.

3.4 Multi-grid

Multi-grid (MG) was introduced in the 1960s by Fedorenko [35] and Bakhvalov [5] to solve partial differential equations. It got full attention in the 1980s, but only recently it has been applied and implemented for N-body problems [13, 107]. MG scales as $\mathcal{O}(N)$.

MG imposes a hierarchical separation of spatial scales. The pair-wise interactions are *split* into a local and a smooth part. The local part are the short-range interactions, which are computed directly. The smooth part represents the slowly varying energy contributions, approximated with fewer terms – *coarsening*. MG uses gridded interpolation for both the charges (source) and the potentials (destination) to represent its smooth – *coarse* – part. The splitting and coarsening are applied recursively and define a grid hierarchy (Figure 4).

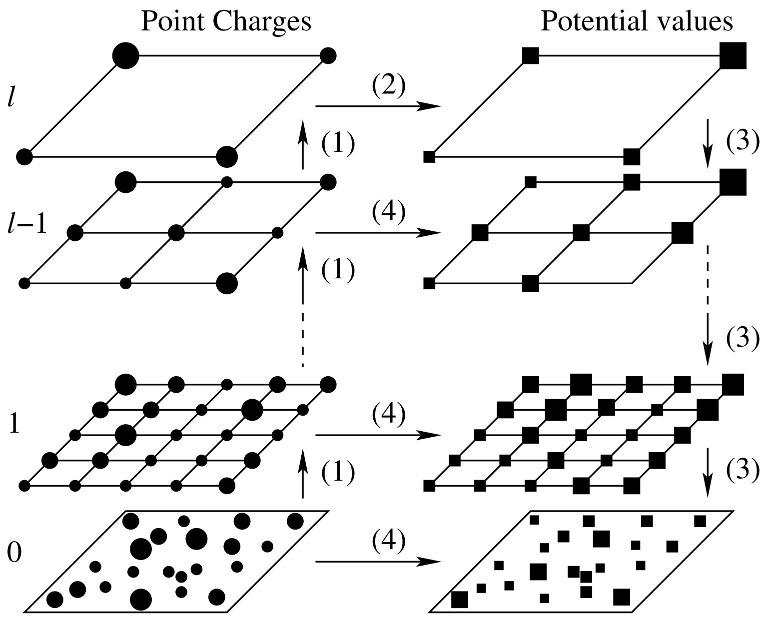


Figure 4: The multilevel scheme of the multi-grid algorithm. (1) Aggregate to coarser grids; (2) Compute potential induced by the coarsest grid; (3) Interpolate potential values from coarser grids; (4) Local corrections.

For the electrostatic potential, the *kernel* is defined by $G(r) = r^{-1}$ and $r = ||\vec{y} - \vec{x}||$. $G(r)$ is obviously not bounded for small r. The interpolation imposes smoothness to bound its interpolation error. By separation, the *smoothed kernel* (smooth part) for grid level $k \in \{1, 2, \ldots, l\}$ is defined as

$$G^k_{\text{smooth}}(\vec{x}_i, \vec{x}_j) = \begin{cases} G_{s_k}(||\vec{x}_j - \vec{x}_i||) & : \quad ||\vec{x}_j - \vec{x}_i|| < s_k \\ G(||\vec{x}_j - \vec{x}_i||) & : \quad \text{otherwise} \end{cases} . \tag{29}$$

Here, s_k is the softening distance at level k and $G_{s_k}(r)$ is the *smoothing function* with $G_{s_k}(s_k) = G(s_k)$. We define $s_k = a^{k-1}s$, typically $a = 2$, together with a corresponding constant sequence of levels with coarsening ratios $1 : a$. G_{s_k} are defined by a polynomial parameterized by s_k (Figure 5) with C^n-continuity. Note that s_k and the coarsening ratios can be defined more general, but it may influence performance and accuracy, especially if s_k/h_k are not of the same order [13, pp. 6-31].

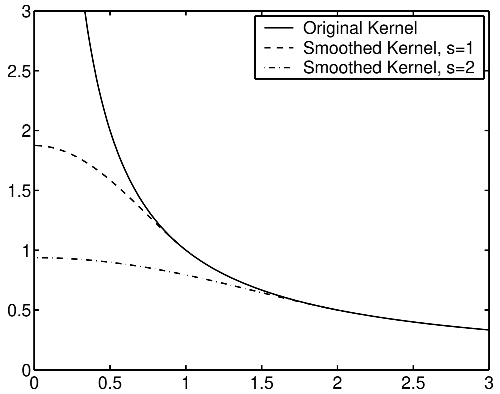


Figure 5: A plot of the original kernel r^{-1} and two smoothed kernels $G^k_{\text{smooth}}(\vec{x}_i, \vec{x}_j)$ with softening distances $s = 1$ and $s = 2$. $G_s(r) = \frac{1}{s}\left(\frac{15}{8} - \frac{5}{4}\left(\frac{r}{s}\right)^2 + \frac{3}{8}\left(\frac{r}{s}\right)^4\right)$, $G_s(r)$ is C^2-continuous.

The corrections of the potentials are required when the modified, smoothed kernel is used instead of the exact one. The correction of the kernel (local part) at level k is given by

$$G^0_{\text{corr}}(\vec{x}_i, \vec{x}_j) = \begin{cases} G(r) - G_{s_0}(r) & : \quad r < s_0, r = ||\vec{x}_j - \vec{x}_i|| \\ 0 & : \quad \text{otherwise} \end{cases} \tag{30}$$

$$G^k_{\text{corr}}(\vec{x}_i, \vec{x}_j) = \begin{cases} G^k_{\text{smooth}}(r) - G_{s_k}(r) & : \quad r < s_k, r = ||\vec{x}_j - \vec{x}_i|| \\ 0 & : \quad \text{otherwise} \end{cases} . \tag{31}$$

Note that $G^k_{\text{corr}}(\vec{x}_i, \vec{x}_j), k > 1$, can be pre-computed and represented by a single table with the corresponding pre-factor $a^{-(k-1)}$ imposing a constant coarsening ratio $1 : a$, $a \in \mathbb{N}$, $a > 1$ and $s_k = a^{-(k-1)}$.

We define an interpolation $I^{k+1} : V^{k+1} \to V^k$ of order p with local support, where V^k represents the potential values at grid level k, and V^0 the particle potentials. The anterpolation (or adjoint interpolation) is defined by $A^k : Q^k \to Q^{k+1}$, where Q^k are the charges at grid level k and Q^0 are the particle charges. X^k are the lattice or particle positions ($X^0 = \{\vec{x}_i\}$). We can then describe a recursive MG scheme with l levels by

$$
\begin{aligned}
Q^{k+1} &= A^k(Q^k) && (32)\\
V^l &= G^k_{\text{smooth}}(X^k, X^k) \cdot Q(X^k) && (33)\\
V^k &= I^{k+1}(V^{k+1}) + G^k_{\text{corr}}(X^k, X^k) \cdot Q(X^k) && (34)\\
E &= \frac{1}{2} V^1 \cdot Q^1 && (35)\\
\vec{F}_i &= \sum_{X^1_j \in \mathcal{X}(\vec{x}_i)} q_i \nabla_j V^1(X^1_j). && (36)
\end{aligned}
$$

$\mathcal{X}(\vec{x}_i)$ denotes the lattice points where $\vec{x}_i$ has local support from I^1. By reformulating Eqs. (32-36), the MG scheme can be described as in Algorithm 4, defining a *V-cycle*. The *V-cycle* reflects the order in which the grids are used; the algorithm telescopes down to the coarsest grid, and then works its way back to the finest grid, describing a *V*. The pseudo-code for **main** handles the aggregation from the interpolation to the particles and computes the force contributions and the total energy. **multiscale** recursively performs the aggregation of charges, the interpolation of potential values and the local correction as depicted in Figure 4. Due to the uniform grids, the interpolation coefficients can be pre-calculated and represented by one 1-dimensional set of coefficients. The same holds for the local correction, since the softening distance is proportional to its corresponding mesh size.

Assuming uniform grids, a constant coarsening ratio of $1 : 2$, a p-order interpolation, h the mesh size of the finest grid and an average particle density $\rho^3 = \frac{N}{V}$

main:

1. anterpolate position charges to the finest charge grid(1) – step (1);
2. call **multiscale**(maxLevel, level 1);
3. interpolate finest potential grid(1) to the position potentials – step (3);
4. correct position potentials – step (4), Eq. (30);
5. compute forces and total energy – Eqs. (35-36);

multiscale(maxLevel, level k):

1. if maxLevel = k then
   (a) compute potential values on coarsest grid(maxLevel) – step (2), Eq. (29);
2. otherwise
   (a) anterpolate charge grid(k) to coarser charge grid(k+1) – step (1);
   (b) call **multiscale**(maxLevel, k+1);
   (c) interpolate coarser potential grid(k+1) to potential grid(k) – step (3);
   (d) correct potential grid(k) – step (4), Eq. (31);

Algorithm 4: Pseudo-code of a recursive multi-grid scheme with V-cycle.

bounded by some constant, the work at particle level and for level k is given by

$$
\begin{aligned}
a_P^0 &= C_P \frac{1}{2} N \cdot \frac{4}{3} \pi s^3 \rho^3 && (37)\\
a_G^k &= C_G \frac{1}{2} \frac{V}{(2^{k-1}h)^3} \cdot \frac{4}{3} \pi (2^{k-1}s)^3 (2^{k-1}h)^{-3} && (38)\\
a_I^0, a_A^0 &= C_I N p^3 && (39)\\
a_I^k, a_A^k &= C_I \frac{V}{(2^{k-1}h)^3} p \,. && (40)
\end{aligned}
$$

Here, a_P^0 denotes the work for the local correction at the particle level, a_G^k is the work for local corrections at the grid level or the potential evaluation for the coarsest grid. a_I^0, a_A^0, a_I^k, and a_A^k represent the interpolation and aggregation work. C_P, C_G and C_I are constants to weigh the different work. For $l \to \infty$, the total work is

$$a_P^0 + \sum_{k=1}^{l} a_G^k + \sum_{k=0}^{l} (a_I^k + a_A^k) = \frac{2}{3}\pi N(\rho s)^3 \left(C_P + C_G \frac{64}{7} (h\rho)^{-6} \right) \tag{41}$$
$$+ \quad 2C_I N p \left(p^2 + \frac{8}{7} \frac{1}{(h\rho)^3} p \right).$$

Thus, the total work is bounded by $\mathcal{O}(N)$. Balancing the work of a_P^0 and $\sum a_G^k$ leads to the choice

$$h = \frac{2}{\rho} \sqrt[6]{\frac{C_G}{7C_P}}, \tag{42}$$

which is independent of the softening distance s. In general, we can deduce $h = \mathcal{O}(\frac{1}{\rho})$, whereas the work of $\sum a_I^k + a_A^k$ is dominated by the terms a_I^0 and a_A^0 and is of order $\mathcal{O}(Np^3)$, essentially the interpolation between particle positions and the finest grid. Finally, the total work is given by

$$\mathcal{O} \left(N \left(\left(\frac{s}{h} \right)^3 + p^3 \right) \right) . \tag{43}$$

The accuracy is governed by the size of the interpolation error of the smooth part. The interpolation error depends on the smoothness of the kernel G, the mesh size h, and the interpolation order p. Assuming a C^n-continuous kernel and at least n order accurate interpolation, the relative force error of the smooth part is $O\left(\frac{h^p}{s^p}\right)$. Thus, the total relative force error is

$$\mathcal{O} \left(\frac{h^p}{\rho^2 s^{p+2}} \right) . \tag{44}$$

A detailed error estimation of MG is given in [107].

In [13, pp. 6-31], the performance is compared with the direct method for a 2-dimensional system of charges and dipoles. In [107], MG is compared against the fast multi-pole method that is implemented in the parallel program DPMTA [95] for water systems. As expected, experiments from [107] show that for $s \to \infty$, MG converges to the direct method and the error drops to zero monotonously. The work increases monotonously with s, until s is large enough to encompass all pairs and remains constant. Furthermore, it was indicated in [107] that MG has stable dynamics for much lower accuracy than multi-pole methods.

3.5 Multi-pole methods

The multi-pole method can be divided into two key components. First, the short-range interactions are computed directly. Second, individual particles interact with groups of distant particles represented by multi-pole expansions. The multi-pole expansion expresses the interaction between a collection of particles and an individual, well-separated particle. Well-separated means that the distance to the sphere enclosing the distant particles is larger than the radius. The subdivision into groups uses a hierarchical spatial decomposition (typically an oct-tree). A multi-pole expansion is calculated for each sub-cell at the finest level representing the effect to distant particles. These expansions are aggregated hierarchically to represent larger and larger groups of particles (upward pass). Then the particles interact with sufficiently far apart (well-separated) sub-cells, covering the whole domain. Thus, ever-larger sub-cells can interact through individual multi-pole expansions at increasing distance. A first practical algorithm (Barnes-Hut) with work complexity $\mathcal{O}(N \log N)$ was applied to astrophysical simulations [6].

Greengard and Rokhlin [48] introduced a local expansion reducing the work complexity from $\mathcal{O}(N \log N)$ to $\mathcal{O}(N)$. This is known as the fast multi-pole method (FMM). The key idea of the local expansion is the interaction between two well-separated groups of particles. The contribution due to another group of particles that is sufficiently far apart is computed by a Taylor (local) expansion utilizing the multi-pole expansion of the distant sub-cell (local expansion). Then the contributions are passed from parent cells to the children sub-cells (downward pass).

FMM has been implemented for different MD applications [11, 48, 73] and is especially suited for parallelization [93, 95, 96, 123]. However, optimized FMM codes tend to be rather elaborate. More importantly, they do not conserve energy during MD simulations unless enforcing unusual high accuracy, e.g., 12th-order multi-poles [10, 128]. Typically, the electrostatic problem comes with usually small distributions of mono-pole moments that cause self cancellation; positive and negative charges are roughly canceled.

3.6 Other fast electrostatic methods

There exist many fast electrostatic methods and implementations. Typically, they are recombinations of these well-known methods or they recast parts in yet other forms that are evaluated using other, general numerical methods.

For example, in [29] the real-space term of the standard Ewald method (first term of Eq. (24)) is evaluated by multi-pole approximation using a tree code and recursion. The multi-pole approximation method based on Taylor expansion of order p scales as $\mathcal{O}(N)$. The authors claim a relative average force error in Eq. (47) of 10^{-5} with a Taylor expansion of order 10. The trade-off is about $N = 15000$ compared to the standard Ewald method.

The Fast-Fourier Poisson (FFP) method [126] ($\mathcal{O}(N \log N)$) implements the reciprocal part by the solution of Poisson's equation in periodic boundary conditions, with Gaussian charge densities as sources. Poisson's equation is solved directly using FFT. The advantage is that the energy part can be evaluated directly in the reciprocal space thereby avoiding a reverse FFT. The forces require only a local summation over the grid since the density of the Gaussian screening function is highly localized. In [100], the Poisson equation is solved by a multi-grid approach ($\mathcal{O}(N)$).

The particle-particle particle-mesh/multi-pole expansion [104] (P^3M/PME, $\mathcal{O}(N \log N)$) is basically an extension of the P^3M [54] method for periodic boundary conditions using multi-pole expansions. The cell multi-pole method [28] (CMM, $\mathcal{O}(N)$) shares the key features of the multi-pole methods, but it uses Cartesian coordinates only, unlike the other formulations that use spherical harmonics. The reduced cell multi-pole method [28] (RCMM, $\mathcal{O}(N)$) is the extension of CMM for periodic systems, where the infinite periodic lattice is handled by the standard Ewald method. The macroscopic multi-pole method [73] (MMM, $\mathcal{O}(N)$) is based on FMM techniques to treat periodic systems. A more detailed summary is given in [117].

Finally, we mention that there exist several implementations of fast electrostatic methods that are especially adapted for multiple time stepping integrator schemes [7, 8, 10, 68, 72, 129].

4 Parallelization

A typical MD simulation is described by Algorithm 1. The work required for the numerical integration is of order $\mathcal{O}(N)$, which can be carried out independently for each particle. The calculation of forces scales as $\mathcal{O}(N^2)$, due to the dominating non-bonded pair-wise interactions in Eqs. (9) and (10). The force evaluation is therefore the most computationally expensive part. Even for $\mathcal{O}(N)$ interactions the forces typically dominate, since the integration is relatively cheap to carry out. Nevertheless, the terms of the sum of interactions $U(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)$ are independent. Altogether, this makes MD – to a certain extent – inherently parallel.This has been exploited by several parallel MD programs [4, 14, 22, 23, 32, 53, 77, 81, 86, 91, 98, 110, 124, 125].

When parallelizing MD, there are two important considerations to make. First, the MD program must perform well for a small number of particles, i.e., less than 1000. There are several interests to carry out simulations with a few thousand particles over a long time scale, e.g., a protein folding simulation with integration time step of 1 fs (10^{-15} s) over 1 ms (10^{-3} s). Second, the MD algorithms should scale to be able to exploit parallel machines.

To meet the computational demands, there are several sequential optimization approaches – the first optimization step. The approaches can be classified into advanced integration schemes (i.e., MTS, p. 9) and fast force evaluation algorithms handling the expensive non-bonded forces (Section 3). Typically, these techniques are combined [129]. For the parallelization, there are five basic strategies [39, 109]:

- Cloning simply assigns each independent simulation to one single processor. This approach is very efficient (no communication) and easy to implement, but its application area is rather limited (e.g., ensemble weather forecast).

- A master-slave approach allocates work among slaves. It has both communication and load balancing difficulties, especially for a large number of processors.

- Atom decomposition based on data replication is an easy but memory-expensive approach. It has poor scaling properties due to global communication [91]. Programs using this decomposition include UHGromos [22], Amber [121], CHARMM [15], Moldy [98], and an early version of EGO [31].

Systolic or hypersystolic loop algorithms [78, 114] are a possible remedy to reduce the memory usage and to improve the scaling.

- Force decomposition involves either force matrix or systolic loop methods. It scales better than atom decomposition by reducing communication costs through the use of a block-decomposition, as in LAMMPS [92], CHARMM [55] and PROTOMOL [D], and as discussed in [87].

- Spatial decomposition uses linked lists and scales very well [80], but it is more difficult to implement and extend. It is used in ddgmq [16], SIGMA [53], NAMD2 [66], PMD [125], EulerGromos [22], DMMD [4], and evaluated in [9], SPRINGS (see paper [C]).

4.1 Load balancing

A good load balancing of the computational work in a parallel environment becomes more important for a large number of processors [66, 86] (see Table 1 in paper [D]). There are two main approaches, namely, static and dynamic load balancing.

Static load balancing is usually appropriate when the computational work load defined by the sum of all interactions $U(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)$ can be estimated and distributed equally among the processors without any run-time information – at least for the most expensive parts. In case of a non-bonded force including the whole domain or a bonded force, we can simply distribute the interactions equally among the processors, since interactions are known for the whole simulation duration. Optionally, one may also consider spatial information to reduce cache trashing or communication. For the non-bonded forces with cutoff, one has to distribute pairs of spatial sub-domains containing interaction candidates, since the interacting particle pairs are not statically defined and change during a simulation. The sub-domains are defined by a cell list algorithm [9][97, pp. 47-52] (see also paper [C]). Typically, one assigns groups of sub-domain pairs, where a group contains all essential sub-domain pairs to compute all interactions for one given sub-domain. The approach assumes that each group has a work load of the same order. The advantage of static load balancing is that we do not need explicit communication to (re-)distribute the work, but it generally does not scale for a large number of processors or for small systems.

Dynamic load balancing can be done either on a local or on a global level. In case of local load balancing, neighboring processors try to balance their work among each other. Global balancing requires several iterations to redistribute the work evenly among all processors but leads in general to a better distribution of the work. The work can be either assigned on demand or implicitly with the help of some estimates from previous distributions. The work is split up in the same manner as for static balancing. In case of dynamic assignment, the master distributes work to the next idle slave. The assigned work should be as big as possible to reduce the communication between the master and the slaves, but small enough to balance out the total work among all slaves. One remedy for global synchronization is to use the same estimates for several iterations, thereby combining the advantages both from dynamic and static load balancing. The estimates are generally valid for a longer period, due to the fact that the particle dislocation is small or only local. However, for medium-sized MD purpose and a moderate number of processors, we can handle global load balancing by one master globally, whereas for massively parallel machines we need a hierarchy of several masters.

4.2 Parallel programming paradigms

Today parallel programming techniques have converged to two main paradigms. MPI[15] [41] is a standard for message-passing based parallel programs in C/C++ and Fortran. It supports Flynn's taxonomy [37] of MIMD[16] and provides a wide variety of message-passing functionalities. MPI was already widely used before its initial standard appeared in 1995. It supports point-to-point and global communication based on some transport layer (e.g., Sockets). With MPI-2 [40], one-sided communication and process spawning was introduced. One of the major advantages of MPI is the rich functionality, relatively low threshold of usage and the compliance for both distributed-memory and heterogeneous environments. Most hardware vendors provide well optimized, native implementations, but not many of these support the full set of MPI-2 extensions.

OPENMP[17] [88] is an application program interface (API) that supports multi-platform shared-memory parallel programming in C/C++ and Fortran. It supports Flynn's taxonomy of SIMD[18]. Its specification is a set of compiler directives, run-time library routines and environment variables that can be used to specify shared memory parallelism. The directives can be seen as "hints" to compilers

[15]**M**essage **P**assing **I**nterface
[16]**M**ultiple **I**nstruction, **M**ultiple **D**ata
[17]**O**pen specifications for **M**ulti **P**rocessing
[18]**S**ingle **I**nstruction, **M**ultiple **D**ata

to parallelize the code. Although OPENMP is a relatively new standard and its first specification appeared in October 1997, it has gained tremendous popularity. One important factor is the low learning threshold and ease to parallelize existing sequential code. The parallelization is fine grained and it scales rather well for a moderate number of processors. It is actually only supported in shared-memory environments, due to the fact that distributed-memory environments have a high latency to access non-local memory, which is problematic for fine grained parallelism. Furthermore, the memory cannot be controlled explicitly, such that code and data may be distributed among different processors.

In the past, several other parallel programming paradigms were used. Today most of them have lost their importance (e.g., PVM [46]) or never really gained wide acceptance among software and hardware developers (e.g., HPF), due to the lack of portability, a high threshold to use (e.g., Pthreads) or exploring only one key requirement.

4.3 Two parallel implementations

Two different parallelization strategies were implemented and evaluated in this thesis work. These are described below.

4.3.1 Spatial decomposition

SPRINGS is an MD program that targets 2-dimensional physical problems (see papers [A] and [B]) with short-range van der Waals interactions in Eq. (10). The simulation region is split into rectangular cells and domains (collection of cells) are distributed among the processors. This is reflected by the multi-cell algorithm,[9],[97, pp. 47-52], organizing the spatial information of particles (see paper [C]). The parallelization is based on MPI using different communication strategies, including the one-sided communication (Figure 6) feature of MPI-2. Furthermore, SPRINGS supports a variety of partitionings of the simulation region to test the communication strategies. Details and results of performance tests with up to 10^7 particles and 128 processors are presented in publication [C].

4.3.2 Incremental parallelization approach

PROTOMOL is a general-purpose component-based MD framework based on incremental parallelization [D]. The approach encapsulates the parallelization details and provides mechanisms to postpone the development of a parallel imple-

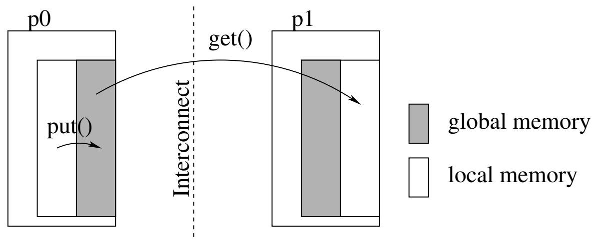


Figure 6: Data exchange based on one-sided communication. (1) put(): processor p0 'exposes' its data to other processors; (2) get(): processor p1 'grabs' needed data independently of p0.

mentation, even in a parallel environment. With co-existing parallel and sequential implementations, we can transparently benefit from already parallelized parts in the framework, and easily experiment with new methods by inserting sequential code.

The parallelization itself is based on range computation with a master-slave concept. The slaves are assigned on demand a range (work-command) by the master, which represents a collection of terms of the sum of all interactions $U(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)$. The splitting of each parallel force is defined and implemented individually, such that each force provides splitting (or distribution) information for the master based on its own characteristic and the actual system. The work defined by the range should be as big as possible to reduced the communication, but small enough to balance out the total work for one time step. The framework assumes that each range of a single force represents computational work of the same order. In some cases, a complete force type is assigned to one slave, i.e., when no parallel implementation is provided. After calculating all interactions (completion of step 2b in Algorithm 1) the force and energy contributions are globally updated by the framework. The initial implementation is based on MPI and data replication. This may not scale well for a large number of processors. Nevertheless, it is proven to perform well for a moderate number of processors, i.e., up to 32. Details and results are presented in publication [D].

5 The PROTOMOL Framework

MD programs may substantially differ in design or level of implementation, but they are essentially all based on Algorithm 1. The proposed MD component-based framework distinguishes PROTOMOL [63] from other well-known MD programs, such as GROMOS [119], Amber [121], CHARMM [15]. A program with similar goals, which provided the initial idea behind PROTOMOL, is NAMD2 [66]. However, NAMD2's design goal is primarily high scalability.

By revisiting and learning from these MD programs, three main requirements were addressed during the design of the framework:

1. Allow end-users to compose integrators and force evaluation methods dynamically. This allows users to *experiment* with different *integration schemes*.

2. Allow developers to easily integrate and evaluate *novel force algorithms* and integration schemes.

3. Develop an encapsulated parallelization approach, where sequential and parallel components co-exist.

Easily customized forces and interchangeable integration schemes have emerged as important requirements. For the last decade, several new ways of solving Newton's equation of motion based on multiple time stepping (MTS) have been developed that improved performance dramatically [7, 39, 60, 57, 79, 129]. It is usually necessary to carefully tune all parts of the MTS scheme to get the full benefits of this technique. Thus, to experiment and to evaluate new and different integration schemes, their dynamic composition and configuration is essential, rather than to change the source code and recompile it. This is discussed in more detail in Section 5.3.

For the developer, it is important to provide an experimental platform and a design of loosely coupled components in which one can easily add novel force algorithms or define and implement new potentials in an orthogonal way. This means also that algorithms can share optimizations common to all of them. Such a framework will hopefully in the future also serve as a platform for fair comparison of different algorithms.

A major problem of most parallel MD applications is that the developer is forced to take into account parallelism issues when modifying the code. This is inappropriate when only implementing and testing experimental ideas. Thus, parallelism must be encapsulated and we must allow that both sequential and parallel implementations can co-exist. Therefore, a design with an incremental parallelization approach is proposed. It suggests that the application developer may start with a sequential implementation of the algorithm, while transparently benefiting from already parallelized parts in the framework. At a later stage one may parallelize the whole new algorithm. The parallelism itself is based on range computation with a master-slave concept. More details are discussed in paper [D]. Section 5.8.2 gives a practical example of how an existing sequential force can be parallelized.

On the other hand, the design and development are also user and application driven:

1. Components to implement sophisticated integrators and fast force evaluators, such as MOLLY [44, 45, 58, 59, 60, 62] and Dissipative Molly [82], and multi-grid summation [13, 107], PME methods [26, 27, 33, 54], and standard Ewald summation [34].

2. Facilities to compare accuracy of force methods, and to make decisions concerning optimal parameters and method for a given accuracy and system.

3. Components to time performance and analyze parallel scalability.

4. Ability to incorporate methods to perform general simulation based on Newton's equation of motion, e.g., the solar system.

5.1 Object-oriented vs. traditional procedural programming

The use of object-oriented programming (OOP) compared to traditional procedural approaches has a trade-off when it comes to run-time efficiency. The OOP introduces some language construct-dependent overhead, which varies strongly on the software design and the actual implementation of the problem. C++ and to some extend Fortran 90 are examples of languages with built-in support of the OOP paradigm. Both languages have inherited their procedural part from their ancestors C and Fortran 77, respectively. C++ supports generic programming (e.g., templates) and (multiple) inheritance enabling great flexibility and ease of code reuse. C++ is a rich and popular OOP language because the programmer can combine high-level OOP abstraction and optimize at low-level machine details, if needed.

However, run-time performance of C++ implementation compared to Fortran 77 or C remains an open question. We did some comparisons on a SGI Origin2000

(R10000) of a specialized procedural Fortran MD implementation [50] against the general C++ MD framework PROTOMOL. We observed that the Fortran code was approximately twice as fast as PROTOMOL, both codes were compiled with MIPSpro compilers with the same optimization options. One important factor for this may be the lack of fully featured optimization compilers for C++, which is likely due to the difficulty to optimize OOP implementations. C statements are mapped linearly to sequences of assembly statements of more or less constant size. C++ breaks this correspondence due to the high non-uniform translation of C++ statements to assembly code. Furthermore, compilers producing highly optimized code for new hardware were first released for Fortran, especially in the high-performance computing segment. Some vendors did not consider C++ as an option for high-performance computing and gave low priority to the development of a C++ compiler.

Despite the run-time efficiency problem, C++ is a strong competitor when correctly designing and implementing complex software. In the long run, flexibility and code reuse will prevail [25, 85], rather than having to continuously re-implement or adapt the software of interest. Thus, the framework C++ was preferred over Java and other object-oriented (OO) languages because of the wide acceptance and the abilities – as discussed before – to combine low-level optimization and high-level abstraction.

5.2 A component-based framework design

There are many different approaches to design software packages, but to satisfy the requirements of flexibility and ease of code reuse, we do not only need a fine grained collection of classes, but also high-level components to easily compose a solution for the problem of interest. This idea is captured by an object-oriented *component-based framework* design (Figure 7) and satisfies also the requirements on page 31. In a component-based framework (or component architecture), a number of modules, interfaces among them, and basic communication services are defined. Examples of component architectures are Microsoft's Component Object Model (COM, cf. [20]) and CORBA [24]. A component architecture for high performance software is the Common Component Architecture (CCA) [19]. For the design of the component-based framework PROTOMOL, three different modules were identified:

1. The front-end provides *components* to compose and configure MD applications. The components are responsible to compose and create the actual

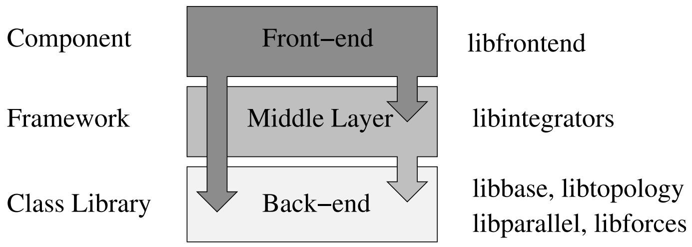


Figure 7: The component-based framework PROTOMOL.

MD simulation set up with its integration scheme and particle configuration, especially interfacing the user. This layer is strongly decoupled from the rest to the extent that the front-end can be replaced by a scripting language. For more information on these topics, see the MD application programmers interface MDAPI [51], and the interactive and steered MD interfaces described in [56, 111].

2. The middle layer is a *white-box framework* for numerical integration reflecting a general MTS design.

3. The back-end is a *class library* carrying out the force computation and providing basic functionalities. It has a strong emphasis on run-time efficiency. In Sections 5.4.2-5.4.5, different options for the force design are evaluated.

The design was also motivated by the goal of encapsulating optimizations such as parallelism without destroying performance. The most important optimizations considered were incremental and scalable parallelization of force evaluation, cell list algorithms to retrieve spatial particle information, and fully customizable non-bonded forces using arbitrary shaped cells and boundary conditions. This is discussed in more detail in Section 5.4.

The discussion of the framework has a strong emphasis on the design of force algorithms, since considerable time was spent to design and implement new force algorithms (e.g., standard Ewald summation, PME, MG, etc.). The front-end is mainly the pre- and post-processing in Algorithm 1 and is not detailed in this thesis, whereas the middle layer is shortly explained to give an overview of the collaboration of integrators and their associated forces.

5.3 Integrators

The integrators represent the middle layer of the framework and are in charge of numerical integration. The collaboration of an integrator object with its associated force objects is based on composition and encapsulation (Figure 8).

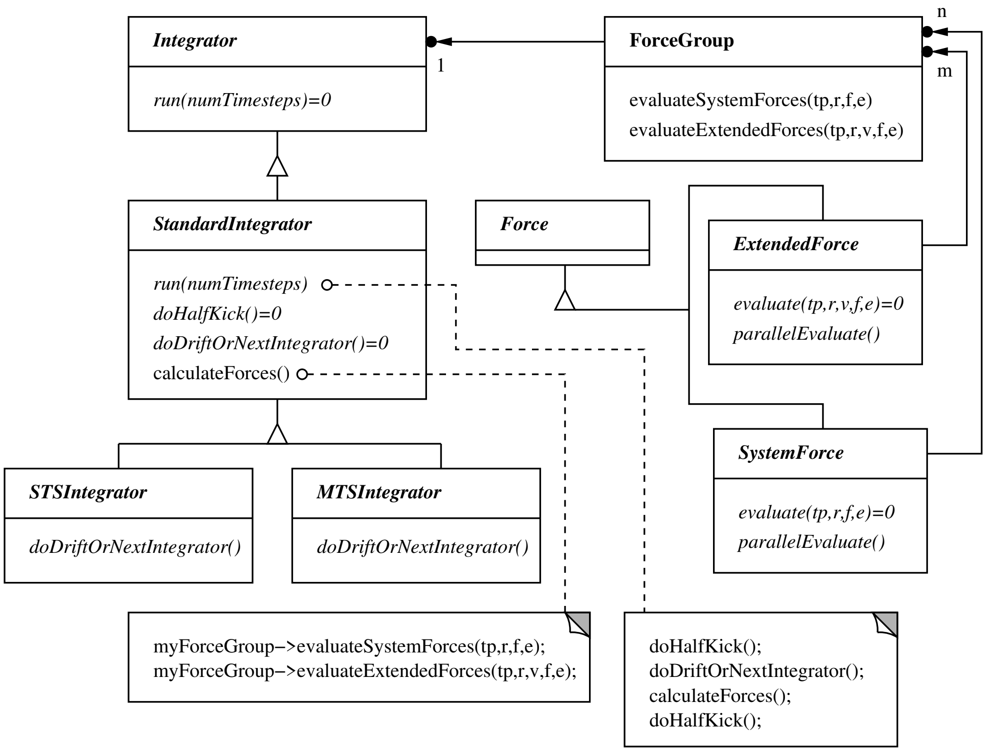


Figure 8: Collaboration diagram between the integrator object and the force objects.

To meet the requirements of dynamically configurable multiple stepping algorithms with arbitrary levels and arbitrary associated forces at each level, the approach of a chain of integrators was chosen (Figure 9). A chain of arbitrary length is defined by recursion, where a single time stepping (STS) integrator terminates the recursion. For the design of integrators, inheritance is used, where MTS and STS integrators are two different main branches implementing Algorithm 1. Inheritance is well-suited to capture the unique and shared behaviors of integrators. Furthermore, it gives great flexibility and has a negligible impact on run-time efficiency in the whole framework. Researchers at the University of Notre Dame have designed and developed the integrator layer in PROTOMOL [61].

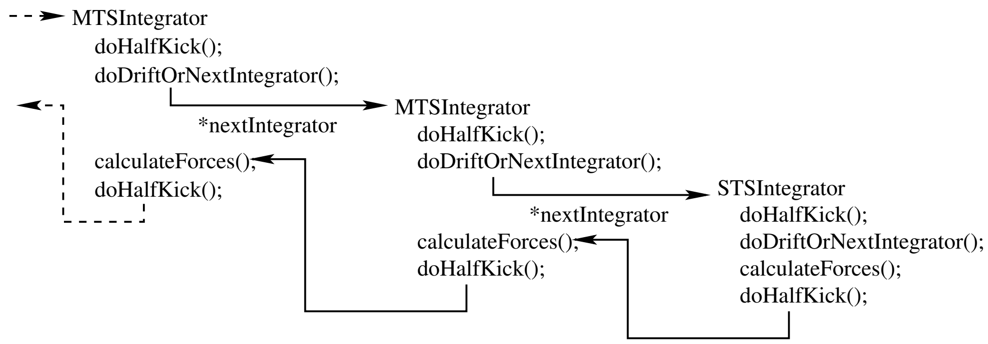


Figure 9: Chain of integrators implementing multiple time stepping schemes. MTS differs from STS in that it calls recursively the next inner integrator before evaluating its forces. The recursion is terminated by an STS integrator.

The associated forces are encapsulated in `ForceGroup` to decouple them from the integrator when invoked to evaluate the forces. This makes it possible to perform some pre- and post-processing, e.g., to distribute the force computation among different processors. Furthermore, in a parallel environment this class handles parallelism and interacts with the parallel library of the framework. The class invokes the actual distribution of the work and updates of force and energy contributions among the nodes. However, this (default) behavior may be overwritten and the parallelism moved up to the integrator level, which corresponds to an optimized parallel update of positions and velocities.

5.3.1 Integrator definition language

At the highest level, the framework defines an integrator definition language (Program 1), in which the user can compose his or her own integration scheme. The grammar imposes the proposed hierarchy (or chain) of integrators, where each level is defined by an integrator of choice and its associated forces. An integrator can be either MTS or STS. The inner most integration (level 0) is defined by STS and MTS for the superior levels. The program fragment 2 defines a simple Leap-Frog integration scheme with a time step of 1 fs and its associated bond, angle, dihedral and improper forces. Program 3 illustrates a three-level MTS integration. The inner most integration uses Leap-Frog, the superior levels use Verlet-I/r-RESPA. It evaluates forces at different time scales. The actual frequency of a force at a given level is recursively defined by the product of the actual cycle length and the frequency of the next inner integrator.

Integrator {
  level **N-1 integrator_name** { # MTS
    cyclelength **value**
    force **forcename** *forceoptions*, [**forcename** *forceoptions*]
    . . .   }
  . . .
  level **0 integrator_name** { # STS
    timestep **value**
    force **forcename** *forceoptions*, [**forcename** *forceoptions*]
    . . .   }
}

Program 1: Grammar for PROTOMOL's integrator definition language.

```
Integrator {
  level 0 Leapfrog {
    timestep 1.0 # [fs]
    force Bond, Angle, Dihedral, Improper}
}
```

Program 2: Definition of Leap-Frog integrator with 1 fs time step and bond, angle, dihedral and improper forces.

```
Integrator {
  level 2 Impulse { # Long-range electrostatics
    cyclelength 4
    force Coulomb -algorithm Full -switchingFunction ComplementSWC1}
  level 1 Impulse { # Medium varying forces
    cyclelength 2
    force Improper, Dihedral
    force Coulomb -algorithm Cutoff -switchingFunction SWC1
    force LennardJones -algorithm Cutoff -switchingFunction SWC1}
  level 0 Leapfrog { # Fast varying forces
    timestep 1 # [fs]
    force Bond, Angle}}
```

Program 3: Three-level Verlet-I/r-RESPA MTS. The fast, medium and slow forces are evaluated at 1 fs, 2 (=1·2) fs, and 8 (=1·2·4) fs, respectively.

5.4 Forces

The forces are designed as separate components and part of the computational back-end. They represent one particular force (a bullet in Algorithm 3) and part of integrators by composition (Figure 8). From an MD modeling point of view and from performance considerations, five different requirements (or customizable options) are proposed. These are discussed below.

5.4.1 Force requirements

R1 An algorithm to select an n-tuple of particles to calculate the interaction.

R2 Boundary conditions defining positions and measurement of distances in the system.

R3 A component to retrieve efficiently the spatial information of each particle. This has $\mathcal{O}(1)$ complexity.

R4 A potential defining the force and energy contributions on an n-tuple.

R5 A component to modify the potential (force and energy contributions) to obtain certain properties of the potential, i.e., switching functions.

In order to address these requirements, several design approaches can be chosen. In general, software engineering comes with a rich multiplicity, where one can solve a problem in many different ways. However, there are infinite nuances between wrong and right. To choose a solution out of the combinatorial solution space depends primarily on the problem and future needs and extensions. These may be difficult to point out.

5.4.2 Option 1: All-inclusive interface

A first option is to implement everything under the same umbrella of a do-it-all interface. This may result in a mammoth class, intellectual overhead and inefficiency. Furthermore, trying to implement the force objects would result in implementing all combinations of requirements R1-R5, and an exponential number of classes. Considering only the non-bonded forces in the actual implementation of PROTOMOL with 3 algorithms (R1), 2 boundary conditions (R2), 1 cell list algorithm (R3), 3 different potentials (R4) and 17 switching functions (R5), would require 306 implementations; without taking into account evaluating simultaneous two different potentials. The same problem occurs when developing a graphical engine for n different geometrical shapes and m different target systems. We would prefer to separate the geometrical shape definition and the target system specific parts than to end up with nm implementations. Moreover, the approach is also extremely rigid, since only one slightly unpredicted customization would make the design useless. The design enforces analysis constraints, but we would like the developers to enforce their own constraints, rather than the design imposing predefined constraints.

5.4.3 Option 2: Multiple inheritance

Another possibility is multiple inheritance, which could avoid the exponential number of classes through a design based on a small number of smart classes implementing each design option. The problem with multiple inheritance is that it can only do a superposition of the classes. This forces the developer to carefully coordinate inheritance and the smart classes. Also, care has to be taken to resolve naming conflicts and inheritance of multiple implementations of the same option. This can result in ambiguous definitions, e.g., simultaneous evaluation of the electrostatic and the van der Waal forces. Another problem of classes based on multiple inheritance is the general lack of complete type information, which may make it impossible for the class to carry out its work directly. Nevertheless, there are well-defined solutions or work-arounds to the relating problems of multiple inheritance, cf. [85, Ch. 15],[113, Ch. 15],[115, pp. 155-167].

5.4.4 Option 3: Generic programming

A third option is generic programming using templates that support meta programming. This is in some way a typed extension to C macros, cf. [103, 105, 113, 120]. Templates enable *parameterization* in ways not supported by regular classes. Template parameters can be seen as place holders that the user will fill out when instantiating an object or defining a type. The code of templated classes is created at compile time depending on user-defined types or values. For example, in the framework design the boundary conditions are type template parameters (periodic or vacuum); the non-bonded force definition takes a boolean parameter indicating whether a switching function is used or not.

Templates also come with options to specialize member functions for one specific instantiation, and to partially specialize templated classes. These features are heavily used in the Standard Template Library (STL); e.g., `copy()` may come with a well-tuned implementation depending on the type of container to copy and the target platform. The drawbacks of templates are that one can neither specialize its data members, nor can one add new data members. Furthermore, one cannot partially specialize member functions. This means that the specialization does not scale like it does for inheritance.

5.4.5 Combined approach: Policy, Strategy, and Traits

The evaluation of advantages and disadvantages of the approaches mention above a combination of templates and inheritance leading to a highly customizable and efficient component design. This is discussed more deeply in [1]. This idea is captured by the Policy pattern[19], better known as the Strategy pattern [42, pp. 315-323]. This pattern promotes the idea to vary the behavior of a class independent of its context. It is well-suited to break up many behaviors with multiple conditions and it decreases the number of conditional statements. Closely related are Traits, which intend to carry some extra information of a class, but without requiring any changes of the class itself. Policies and Traits have a lot in common, but the Policies tend to be behavioral, whereas Traits allow subtyping.

5.4.6 Force design of PROTOMOL

The force design defines one typical type of force (e.g., angular forces in molecules) and has an interface defining all necessary parameters for the computation of force and energy contributions. The design of forces itself uses the Policy pattern [1], as discussed in Section 5.4.5. The algorithm to select the n-tuples (R1) is customized with the rest of the four requirements (R2-R5). Some forces may not allow any parameterization at all since they are specialized and hand-crafted to carry out one particular case, e.g., propagating force contributions from an external device like a haptic device. At first, it seems more natural to customize the potential (R4) with an algorithm (R1), but after analyzing the general application requirements and their dependencies, it makes more sense in terms of design and efficiency to choose the first design approach. This becomes more obvious when we like to evaluate simultaneously different types of forces with the same algorithm. Furthermore, for specialized forces like PME, it would be practically impossible to design a proper solution, since some parts do not rely on the selection of n-tuples but on grid evaluation algorithms (see Figure 19 for all

[19] A pattern describes a solution to a recurring problem that arises in specific design situations.

available forces). In order to illustrate how forces are customized with different policy choices, two examples are given below.

The evaluation of angular forces in molecules is a good example of a specialized force type. Its nature is simple and of static character, which means that the 3-tuples of atoms are given by the initial input structure. The potential energy is defined by (see also Eq. (7))

$$U_m^{\text{angle}} = \frac{1}{2} K_A \left(\theta_{ijk} - \theta_m\right)^2, \quad \theta_{ijk} = \cos^{-1}\left(\frac{(\vec{x}_j - \vec{x}_i)\cdot(\vec{x}_j - \vec{x}_k)}{||\vec{x}_j - \vec{x}_i||\cdot||\vec{x}_j - \vec{x}_k||}\right). \quad (45)$$

It does not share any behaviors with other forces, besides the boundary conditions defining the measurement of distances to compute the angle θ_{ijk} of the 3 atoms. Program 4 declares the angle force with periodic boundary conditions. Since the boundary condition is a template and known at compile time the compiler can optimize the code extensively, e.g., for vacuum the compiler only applies an ordinary vector difference to compute the distance.

```
Force* aForce = new AngleSystemForce<PBC>();
```

Program 4: Dynamic instantiation of an angular force with periodic boundary conditions.

More complex force objects stem from non-bonded forces. For example, to define an electrostatic force (Program 5), we may choose a cutoff algorithm (R1) that considers only the closest neighboring atoms. To find the closest atoms in constant time, we use a cell manager (`CM`, R3) based on a cell list algorithm. `PBC` defines periodic boundary conditions (R2). `OneAtomPair` defines the energy and force contributions between two arbitrary atoms. Here, `CoulombForce` implements Eq. (9) modified by a C^1-continuous switching function (`C1SwitchingFunction`) using Eqs. (18) and (19). The boolean value defines the switching function should be applied or not, which is for optimization purpose. For example, to compute all interactions over the whole domain no switching function is required and the compiler can drop all code related to the switching function. In case of a MOLLY integrator scheme, `OneAtomPair` would be replaced by `OneAtomPairMollifyLJ` to implement a 'mollification'. Table 5 gives a complete overview of supported policy choices. Figure 24 details `OneAtomPair`, `OneAtomPairMollifyLJ` and other supported pair-wise interactions, e.g., the simultaneous evaluation of two different forces. The actual object creation is a little more complicated, due the numerous policies and usage of nested templates. A generic object creation is discussed in Section 5.5.

```
typdef
OneAtomPair<PBC,CM,C1SwitchingFunction,CoulombForce,true> OneAtomPairType;

class NonbondedCutoffSystemForce<PBC,CM,OneAtomPairType>;
```

Program 5: Declaration of a Coulomb force with cutoff and C^1-continuous switching function.

5.4.7 Force interface

The forces are designed with a common interface, the so-called `evaluate(...)` and optional `parallelEvaluate(...)` method, which does the evaluation of the force contributions based on its parameterization and policy choices.
`evaluate(...)` is a pure virtual function and expects always a sequential implementation. `parallelEvaluate(...)` is an optional (or 'override-able') interface for parallel implementation. In case no parallel implementation is provided, it will call the sequential method `evaluate(...)`. This mechanism allows the developer to postpone a parallel implementation, even in a parallel environment. It reflects the idea of incremental parallelization (see paper [C]).

The force interface can be seen as a function object, but to make it a real functor, the Command pattern [42, pp. 233-242] would be needed to pass the parameters. Note that this would be appropriate if we would need an 'undo' functionality or delayed (lazy) evaluation.

5.4.8 Force subtyping

The force design distinguishes between system forces (Figure 21) and extended forces (Figure 23). The system forces are designed to handle ordinary interactions of the form $U(\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_N)$, whereas the extended forces are designed for velocity dependent interactions. Furthermore, the forces are encapsulated in `ForceGroup`, which is part of the intergrator. The contributions of the system and extended forces are invoked by separate methods from an integrator (Figure 8), such that different pre- and postprocessing for those two force types can be performed. Note that from the user point of view, there is no distinction between extended and system forces.

Beyond this subtyping, the forces are divided between bonded and non-bonded forces. Bonded forces keep local neighbor information that is typically static and does not rely on requirement R3, whereas non-bonded forces operate on dynamic neighborhoods (R3) or may include the whole computational domain. This means that non-bonded forces may be computed using cutoff techniques, or in the full domain. Appendix B describes the force computation design in detail. Figure 19 is an overview of actually supported forces.

5.5 Force factory

Once the forces are designed and implemented, we need to create (or instantiate) the actual force objects needed by integrators. One approach could be to let the integrator or some other function create them. The integrator or the function would have the responsibility to parse the input and instantiate the corresponding

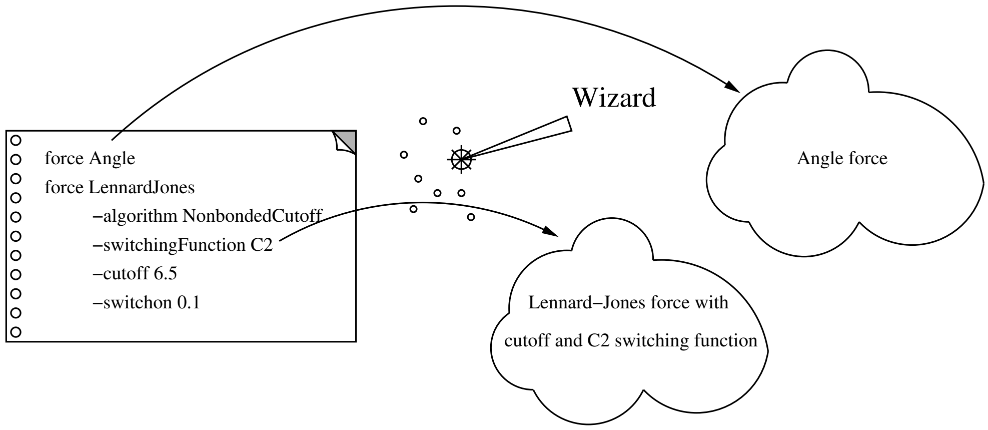


Figure 10: Correspondence between the input definition and the actual force object.

force objects (Figure 10). This would lead to a rigid design, i.e., huge if-then-else or switch-case statements due to the numerous force types. Additionally, it would not only be difficult to add or remove force types, but also the correspondence between the input definition and the actual force object would be hard-coded. We would rather like to make visible force types of interest only and let the forces decide the correspondence. Thus, we need some sort of wizard, that can transform a given input definition into a real force object.

5.5.1 Solution: Abstract Factory and Prototypes

The requirements of object creation are satisfied by the Abstract Factory [42, pp. 87-95] and the Prototype [42, pp. 117-126] patterns. The Abstract Factory pattern delegates the object creation, and the Prototype pattern allows dynamic configuration. The logical steps in using an Abstract Factory and Prototype combined is as follows:

1. **Prototype registration.** The force factory is dynamically configured by registering force prototype objects. A force prototype object is a force object with undefined parameter values. The set of registered force prototype objects defines the force (algorithm) space available to the user or components. This enables us to customize the available forces. For example, we can eliminate all those that do not make sense logically.

```
ForceFactory::registerExemplar(
  new NonbondedCutoffSystemForce<PBC,CM,
         OneAtomPair<PBC,CM,C2SwitchingFunction,LennardJonesForce,true> >());

ForceFactory::registerExemplar(new AngleSystemForce<PBC>());
```

Program 6: Registration of prototypes: angular force and Lennard-Jones force prototype with cutoff and C^2-continuous switching function.

2. **Force prototype symbol table.** Before any force object can be instantiated, some sort of correspondence – map – between the input and the available (or registered) force prototypes must be defined. In C++, one could use Run-Time Type Identification (RTTI), but this feature is non-portable and may lead to ambiguous correspondences. In order to define this map globally and of static type, the forces are responsible to provide a unique identification string, based on the force class and their policies (Table 1). More precisely, force options defining policies are part of the identification string, e.g., interpolation scheme. For forces using nested templates, the identification string is defined as the deep-copy principle, e.g., the complement of a switching function, which expects a switching function as template parameter.

Force	Identification string
Angle	`'Angle'`
LJ	`'LennardJones -algorithm NonbondedCutoff -switchingFunction C1'`

Table 1: Identification string of angle force and a Lennard-Jones force.

3. **Force definition language.** The definition of a force is essentially based on the unique identification string. A force definition starts always with the keyword `force` followed by a keyword (i.e., **forcename**) reflecting the type of force and the rest of the unique identification string (i.e., *forceoptions*). **forcename** may consist of two words to evaluate two forces simultaneously. The **forcename** is followed by parameters of the force (i.e., *forceoptions*) to parameterize the force with the desired values, e.g., cutoff distance. Some force may have additional policies or force parameters, e.g., interpolation scheme or friction coefficient. Other forces are defined only by one keyword, e.g., angle forces. 'compare' and 'time' are for comparison purpose (see Section 5.6).

```
level n integrator_name {
        ...
        force [compare] [time] forcename forceoptions
        [force [compare] [time] forcename forceoptions]
        ...
}
```

Program 7: Grammar for PROTOMOL's force definition language.

4. **Force parsing.** The force factory takes the rule of the wizard. An extended map based on the identification string defines the correspondence. In general, the order of definition does not matter, but more complex force definitions require an exact match in order to preserve uniqueness. Once the correct force prototype is identified, the expected force parameters are retrieved from the prototype. To let the prototype define the parameters makes the parsing general and ties the number and keywords of parameters to the force prototype. More exactly, the parameters are tied to the policies, such that there is only one definition in the whole system that is in the same file as the implementation of the according policy. Furthermore, this allows us to provide default values, if needed. Next, the force factory parses the requested parameter values and passes them with their values to an Abstract Factory method `make(...)` (*Template Method*) to create the desired force. The class-dependent behavior for the Abstract Factory method is implemented in `doMake(...)` (*Hook Method*), which is called by `make(...)`. This technique is described by the Template Method pattern [42, p. 325].

5. **Force object creation.** Finally, a fully featured force object with set parameters is created by the prototype. In order to make the dynamic configuration and the actual object creation independent, and the factory globally accessible, the force factory uses the Singleton pattern [42, pp. 127-134]. This avoids the creation of unnecessary factories, and allows to pre-customize the factory with behavior common to all forces. The latter is used to define boundary conditions and the type of cell manager, since they are unique policies inside the whole MD application. Furthermore, the force factory provides features for lazy registration of prototypes and their destruction.

5.6 Functionalities to compare force algorithms

Since one of the requirements of the framework is the ease to evaluate and compare different force algorithms, functionalities were added to the framework for this purpose (Figure 20). At present, pairs of forces can be compared to determine energy and force errors of new force methods:

$$\mathrm{FE_{max}} = N\frac{\max_i m_i^{-1/2}||\tilde{\vec{F}}_i - \vec{F}_i||}{\sum_i m_i^{-1/2}||\vec{F}_i||} \tag{46}$$

$$\mathrm{FE_{avg}} = \frac{\sum_i m_i^{-1/2}||\tilde{\vec{F}}_i - \vec{F}_i||}{\sum_i m_i^{-1/2}||\vec{F}_i||} \tag{47}$$

$$\mathrm{FE_{abs}} = \max_i ||\tilde{\vec{F}}_i - \vec{F}_i|| \tag{48}$$

$$\mathrm{UE} = \left|\frac{\tilde{U} - U}{U}\right| . \tag{49}$$

Here, $\tilde{\vec{F}}_i$ and $\tilde{U}$ are the contributions of the force method to be evaluated. $\vec{F}_i$ and U denote the values of the reference force method (or the most accurate one), e.g., obtained by direct calculation. The comparison is performed on-the-fly, such that the reference force does not affect the current simulation. For example, one can compare a fast electrostatics method such as PME using two grid sizes, such that the more accurate one serves as an accuracy estimator (Program 8).

For the benchmarking of forces a timer, function was implemented to measure the total and average time spent in dedicated force methods (Program 8). Both comparisons functions can be nested to evaluate accuracy and run-time performance simultaneously.

```
force compare time force Coulomb -algorithm PMEwald -real -reciprocal
        -correction -interpolation BSpline  -cutoff 6.5 -gridsize 10 10 10
force compare time force Coulomb -algorithm PMEwald -real -reciprocal
        -correction -interpolation BSpline  -cutoff 6.5 -gridsize 20 20 20

force time LennardJones -algorithm NonbondedCutoff -switchingFunction C1
                        -cutoff 8.0
```

Program 8: Examples of accuracy and timing comparison.

The comparison of force pairs is based on the Count Proxy pattern [42, pp. 207-217] to link two forces together and calculate the actual errors. The Count Proxy pattern makes comparison transparent from the integrator point of view, such that the contributions of the reference force are not mixed up with the current simulation. To handle both system and extended forces with their different interfaces polymorph inheritance is used. The timer function is based on the same approach as for the accuracy comparison. Due to the transparency, the approach allows the nesting of the accuracy and timing functionalities.

5.7 Implementation and validation of fast electrostatic force algorithms

During this thesis three different fast electrostatic force algorithms were developed and incorporated into the framework. More design details are given in Figure 22. The correctness of implementation of these methods was tested on a 141-molecule water system (Figure 27) with periodic boundary conditions 25Å×25Å×25Å or vacuum. The tests were performed on an IBM p690 Regatta Turbo system. Calculations were performed in 64-bit arithmetic and a machine precision of order 10^{-16}. The following three sections summarize the features of the algorithms.

5.7.1 Standard Ewald summation

The Ewald method is described in Section 3.2. The real, reciprocal and correction terms (the last 3 terms of Eq. (24)) can be evaluated independently. This is typically used in combination with MTS to split the force into fast and slow varying parts. Furthermore, the real space term supports not only simple truncation, but also switching functions to modify the potential. The implementation supports parallelism (range computation) for the real, reciprocal and correction terms. It handles MD systems for both vacuum and periodic boundary conditions. Due to the relatively expensive sine and cosine functions, PROTOMOL uses look-up tables and the addition theorem to evaluate $\cos(\vec{k}\cdot\vec{x}_i)$ and $\sin(\vec{k}\cdot\vec{x}_i)$ more efficiently in Eqs. (24) and (25); erf(x) and erfc(x) are by default not approximated, since the system platforms provide a well optimized implementation.

Figure 11 shows the maximum relative force error of the Ewald method for different accuracy parameters ϵ compared to the Ewald method with accuracy parameter $\epsilon = 10^{-18}$ (see Eqs. (26-28)) under periodic boundary conditions. There was no significant improvement for smaller ϵ, which is obvious, due to machine precision of order 10^{-16}. The maximum relative error of the Ewald implementation compared against the direct method in vacuum is less than 10^{-15} with an accuracy parameter $\epsilon = 10^{-18}$ and a unit MD cell 10^8 times larger than the minimal bounding box of particles. Figure 12 illustrates the corresponding normalized run-time. Figure 13 shows excellent energy conservation with a maximum relative force error of order 10^{-10}. Moldy [98] was also used for further validation.

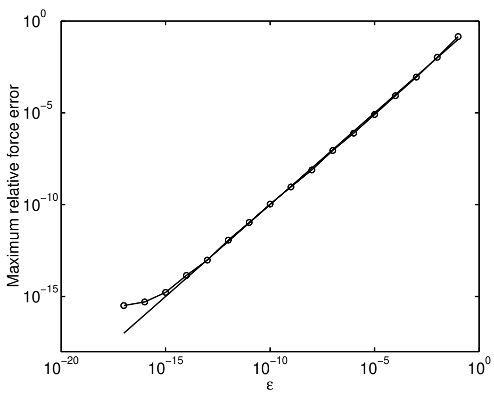


Figure 11: The maximum force error for a given accuracy parameter ϵ.

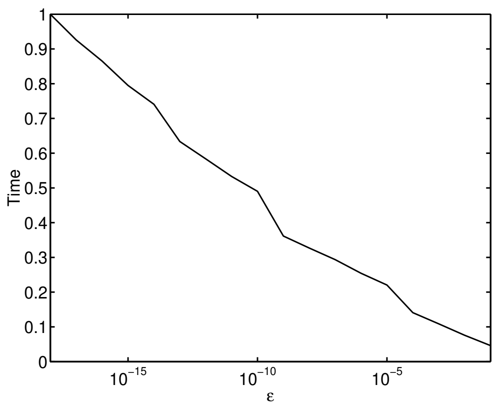


Figure 12: Normalized run-time for a given accuracy parameter ϵ.

5.7.2 Particle-mesh based Ewald summation

PROTOMOL supports PME (Section 3.3) with a generic interpolation scheme interface of arbitrary order. Actually, B-splines and Hermitian interpolation are implemented. The real, reciprocal and correction term can be evaluated separately, as for the standard Ewald implementation. The real term can be modified by a generic switching function. The implementation supports parallelism (range computation) for the real and correction terms. The work of the reciprocal term cannot be distributed in the actual implementation.

The maximum relative error of the PME method compared against the standard Ewald summation is less than $2 \cdot 10^{-14}$. Both methods used an accuracy parameter $\epsilon = 10^{-18}$. The PME was defined with mesh size of 0.1Å, cutoff in real-space part $r_c = 10$, and B-splines of order 12. Figure 13 shows excellent energy conservation with a maximum relative force error of order 10^{-5}. NAMD2 [66] was used to validate the results.

5.7.3 Multi-grid

Multi-grid (Section 3.4) is supported with a generic interpolation scheme interface of arbitrary order. B-splines and Hermitian (cubic and quintic) interpolation are implemented. The direct and smooth part (at particle level) can be evaluated separately. The kernel and its softening function are generic and actually C^1-, C^2-, C^3- and C^4-continuous polynomial smoothing functions for Coulomb interactions are implemented (e.g., G_s in Figure 5). The core of multi-grid handles vacuum or periodicity in each dimension independently. The question of periodic boundary conditions for the coarsest grid evaluation remains an open research question. The actual implementation performs an ordinary matrix-vector multiplication for periodic boundary conditions and vacuum.

The MG code in [107] was used for validation. We obtained the same errors as presented in [107] for the 6848-molecule water system with cubic and quintic Hermitian interpolation. Figure 13 shows the energy conservation for a 141-

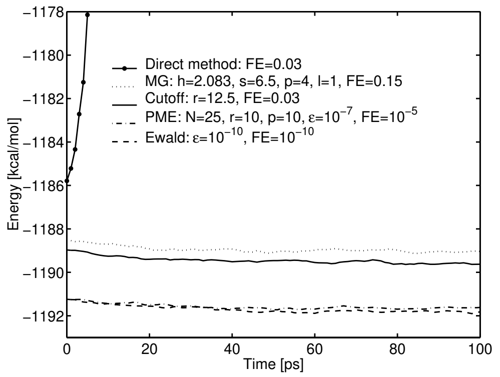


Figure 13: The energy for periodic boundary conditions and step size 1 fs.

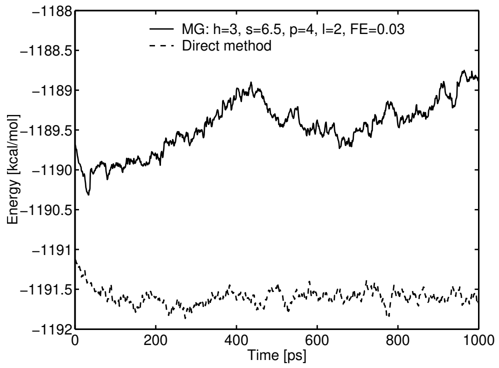


Figure 14: The energy for vacuum and step size 1 fs.

molecule water system with periodic boundary conditions. MG shows excellent energy conservation for low accuracy (maximum relative force error of order 0.15). The direct method does not consider any images, only the original unit MD cell. Using truncation at 12.5 Å (half of the unit MD cell dimension) together with a C^1-continuous switching function performs well and has the same time complexity as the direct method. In case of a small cutoff 6.5 Å, the energy fluctuates around -1172.6 kcal/mol (not shown in this figure). Figure 14 illustrates the energy conservation for vacuum. The minor energy drift compared with the direct method is due to translations of the grids following the particles. The reason is that the potential energy function is not invariant under global translation and/or rotation of the particle positions.

5.8 Practical examples of extendibility and optimization

In order to show the extendibility and flexibility, two practical examples are given. The first one shows how to implement a new force, whereas the second example describes the parallelization of a given force.

5.8.1 Adding a new force

Coulomb crystal systems are defined by ions with an additional magnetic trap [52, 94, 102, 116]. A common used trap is the Paul Trap attraction, which is given by

$$U^{\text{PaulTrap}} = \sum_{i=1}^{N} \left(\frac{1}{2} m_i \omega_{xy}^2 (x_i^2 + y_i^2) + \frac{1}{2} m_i \omega_z^2 z_i^2 \right), \quad \vec{r}_i = (x_i, y_i, z_i)^\top. \tag{50}$$

Here, $\vec{r}_i$ is the coordinate of particle i. ω_{xy} and ω_z parameterize the Paul Trap attraction.

The interaction is implemented in the class `PaulTrapExtendedForce` and implements an extended force (`ExtendedForce`). `ExtendedForce` was chosen as base class to be able to incorporate a velocity dependent term in future implementations. The class `PaulTrapExtendedForceBase` is inherited privately and defines only static functionalities, i.e., the keyword `PaulTrap`. The method `evaluate(...)` implements Eq. (50) (Program 9). The first constructor (the empty one) is used when creating a prototype, while the second one is called in the Abstract Factory method `doMake(...)` to instantiate a fully parameterized object with ω_{xy} and ω_z. `getParameters(...)` defines the parameter names and returns also the parameter values. Program 10 registers a prototype of the force and makes the force 'visible'. The complete implementation is given in Appendix C, Program 13.

```
template<class TBoundaryConditions>
class PaulTrapExtendedForce : public ExtendedForce,
                              private PaulTrapExtendedForceBase {
public: // Constructors
  PaulTrapExtendedForce():myOmegaXY(0.0),myOmegaZ(0.0){};
  PaulTrapExtendedForce(Real omeagXY, Real omegaZ):myOmegaXY(omegaXY),
                                                    myOmegaZ(omegaZ){};
public: // From class SystemForce
  virtual void evaluate(...){
    const TBoundaryConditions &boundary = ...;
    Real e = 0.0;
    for(int i=0;i<topo->atoms.size();i++){
      Real c = topo->atomTypes[topo->atoms[i].type].mass*;
      Coordinates pos(boundary.basisPosition((*positions)[i]));
      Coordinates f(c*myOmegaXY*myOmegaXY*pos.x,c*myOmegaXY*myOmegaXY*pos.y,
                    c*myOmegaZ*myOmegaZ*pos.z);
      (*forces)[i] -= f;
      e += 0.5*c*(myOmegaXY*myOmegaXY*(pos.x*pos.x+pos.y*pos.y)+
                  myOmegaZ*myOmegaZ*pos.z*pos.z);
    }
    energies->otherEnergy += e;
  }
public: // From class Force
  virtual void getParameters(vector<ParameterType>& parameters) const{
    parameters.push_back(ParameterType("-omegaXY",VarValType::REAL,
                                       VarVal(myOmegaXY)));
    parameters.push_back(ParameterType("-omegaZ",VarValType::REAL,
                                       VarVal(myOmegaZ)));
  }
protected: // New methods
  virtual Force* doMake(string&, const vector<VarVal>&) const{
    Real omegaXY = values[0].getReal();
    Real omegaZ = values[1].getReal();
    return new PaulTrapExtendedForce(omegaXY,omegaZ);
  }
...
};
```

Program 9: Implementation of a Paul Trap attraction.

```
ForceFactory::registerExemplar(new PaulTrapExtendedForce<PBC>());
```

Program 10: Registration of a Paul Trap attraction.

5.8.2 Parallelizing a force

In order to parallelize a given force, `parallelEvaluate(...)` has to be overwritten. Furthermore, we have to define a splitting of the force to enable range computation. For example, a velocity dependent friction could be implemented as described in Program 11. In this program, `n` is the number of particles, and `blocks` defines the number of blocks, typically the number of processors. The same splitting must be implemented in the method `numberOfBlocks(...)` (see complete implementation in Program 14) to allow the master processor to distribute the work. Whenever a slave enters `parallelEvaluate(...)`, `next()` will be true if the actual block `i` has not been processed by any other slave.

Like for this simple force, we may rather implement the actual force evaluation in a private method `doEvaluate(...,from,to)` with range ability to avoid code duplication. The sequential implementation calls it with the full range, whereas the parallel version calls it with the according block indices (Program 14).

```
virtual evaluate(...){...} // sequential implementation
virtual parallelEvaluate(...){
  ...
  int n =  positions->size();
  int blocks = ...; // number of blocks
  for(int i = 0;i<blocks;i++){
    if(topo->parallel->next()){
      int to = (n*(i+1))/blocks;
      if(to > n) to = n;
      int from = (n*i)/blocks;
      for(int j=from;j<to;j++){
        (*forces)[j] += (*velocities)[j]*myF;
} } } }
```

Program 11: Parallelization of a velocity dependent friction.

5.9 Performance

In the following three sections the run-time performance of PROTOMOL is discussed. We compare PROTOMOL to NAMD2 [66] for sequential and parallel runs, and compare the different fast electrostatic force algorithms of PROTOMOL. The performance evaluations are based on three systems, a 141-molecule water system (Figure 27), BPTI with water (Figure 25), and ApoA1 with water (Figure 28).

5.9.1 Sequential scalability

We performed a sequential comparison of PROTOMOL with NAMD2, since they have similar basic functionalities and are both implemented C++. In order to make the comparisons fair, we used the same integration scheme and considered the same forces. Both configurations use Leap-Frog with time step 1 fs (Program 12). For the structural and initial simulation state, the identical input was used. Table 2 demonstrates the sequential performance of PROTOMOL against NAMD2. One can conclude that PROTOMOL performs well and has a slightly better (sequential) scaling than NAMD2, in respect of problem size. All runs were performed on a dedicated node of a Linux cluster with 1.26 GHz Pentium III processors. PROTOMOL was configured with a cell size of 5 Å for the water case and $3\frac{1}{3}$ Å for the two other cases in order to exploit the cache optimally.

Test case		# atoms	PROTOMOL		NAMD2		Ratio
		N	[s]	[μs] / N	[s]	[μs] / N	
Water	vacuum	423	0.0108	25.53	0.0122	28.84	0.89
	periodic		0.0156	36.88	0.0149	35.23	1.05
BPTI	vacuum	14,281	0.7826	54.8	0.889	62.25	0.88
	periodic		1.386	97.01	1.216	85.15	1.14
ApoA1	vacuum	92,224	5.622	60.96	6.537	70.88	0.86
	periodic		9.259	100.04	10.056	109.04	0.92

Table 2: Sequential comparison of PROTOMOL vs. NAMD2 for one time step in average on a Pentium III, 1.26 GHz running Linux RedHat. All test cases use Leap-Frog integration with 1 fs time step and a non-bonded force cutoff of 10 Å, and Lennard-Jones and Coulomb forces with C^1-continuous and shift switching function, respectively.

```
Integrator {
  level 0 Leapfrog {
        timestep 1.0 # [fs]
    force Improper, Dihedral, Bond, Angle
    force LennardJones Coulomb     # Combining LJ and Coulomb evaluation
        -switchingFunction C2       -switchingFunction Shift
        -algorithm NonbondedCutoff  -switchon 0.1 -cutoff 10.0 }}
```

Program 12: PROTOMOL: Integrator and force definitions used for the performance comparison test. Lennard-Jones and Coulomb contributions are evaluated simultaneously.

5.9.2 Parallel scalability

Figures 15 and 16 illustrate the parallel scalability of PROTOMOL and NAMD2. PROTOMOL scales smoothly and well up to 32 processors. For one and two processors PROTOMOL does static load balancing and dynamic load balancing for three or more processors. For dynamic load balancing, one dedicated processor takes the role of master serving the slaves. Table 3 contains the average time of a sequential run for one time step. PROTOMOL and NAMD2 have comparable sequential performance, PROTOMOL is $5-10\%$ faster for vacuum, but $5-10\%$ slower for periodic boundary conditions than NAMD2. The simulation setup for all test cases was the same as for the sequential comparison, except for a cell size

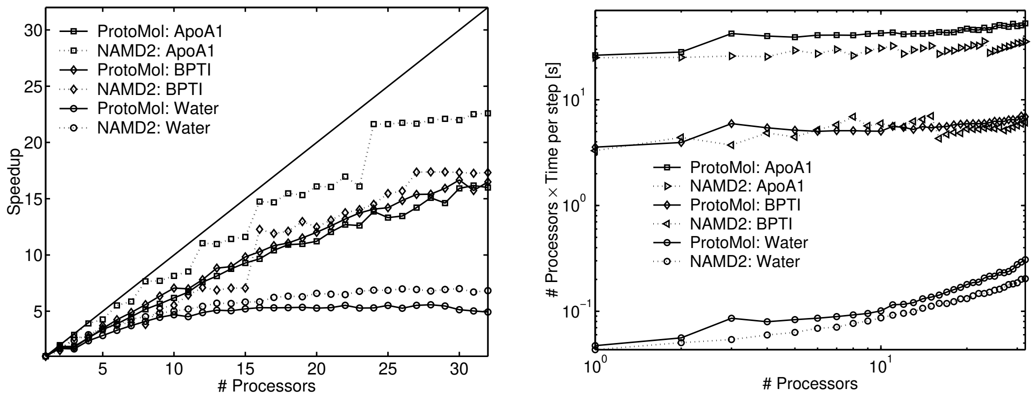


Figure 15: Parallel scalability for periodic boundary conditions.

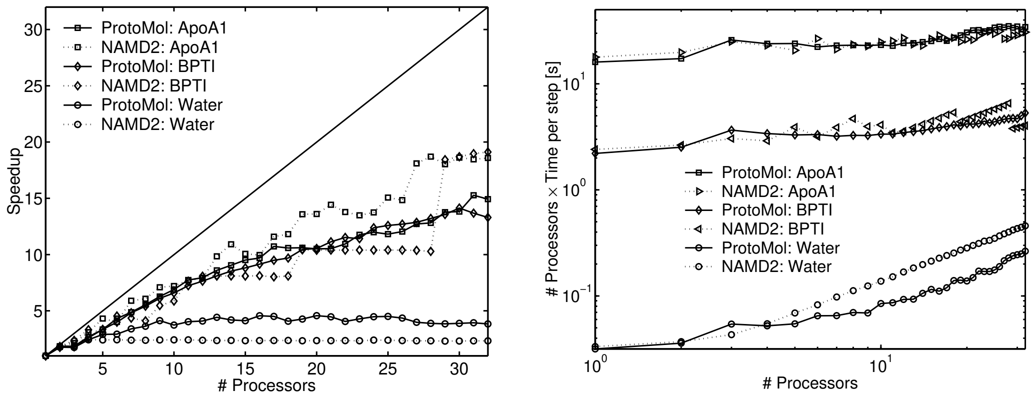


Figure 16: Parallel scalability for vacuum.

of 5 Å for all PROTOMOL runs. All runs were performed on an Origin2000 with 195MHz R10000 processors.

Test case		# atoms N	PROTOMOL [s]	NAMD2 [s]
Water	vacuum	423	0.0317	0.0334
	periodic		0.0473	0.0434
BPTI	vacuum	14,281	2.209	2.408
	periodic		3.562	3.304
ApoA1	vacuum	92,224	16.03	17.83
	periodic		26.36	25.04

Table 3: Sequential comparison of PROTOMOL vs. NAMD2 for one time step in average on an Origin2000 with 195MHz R10000 processors. All test cases use Leap-Frog integration with 1 fs time step and a non-bonded force cutoff of 10 Å, and Lennard-Jones and Coulomb forces with C^1-continuous and shift switching function, respectively.

5.9.3 Scalability of fast electrostatic force algorithms

Figure 17 shows the run-time efficiency of the PME and the standard Ewald method with a maximum relative force error of 10^{-5} for the three test cases with periodic boundary conditions. The Ewald summation with an accuracy parameter ϵ of 10^{-18} served as reference. Figure 18 illustrates the run-time efficiency of MG with a maximum relative force error of 10^{-2} and 10^{-3} compared to the direct method for vacuum. As expected, the direct method scales as $\mathcal{O}(N^2)$ and MG scales as $\mathcal{O}(N)$, which is supported by the observed speedup of 620 or more for a 10^6 Ca^+_{40} Coulomb crystal simulation compared with the direct method and maximum relative force error of $5 \cdot 10^{-3}$. Note that one may get even better performance numbers by extensive fine-tuning of the MG parameters. Table 4 summarizes the performance numbers both for vacuum and periodic boundary conditions. All runes were performed on an IBM p690 Regatta Turbo system in production mode. Calculations were performed in 64-bit arithmetic and a machine precision of order 10^{-16}.

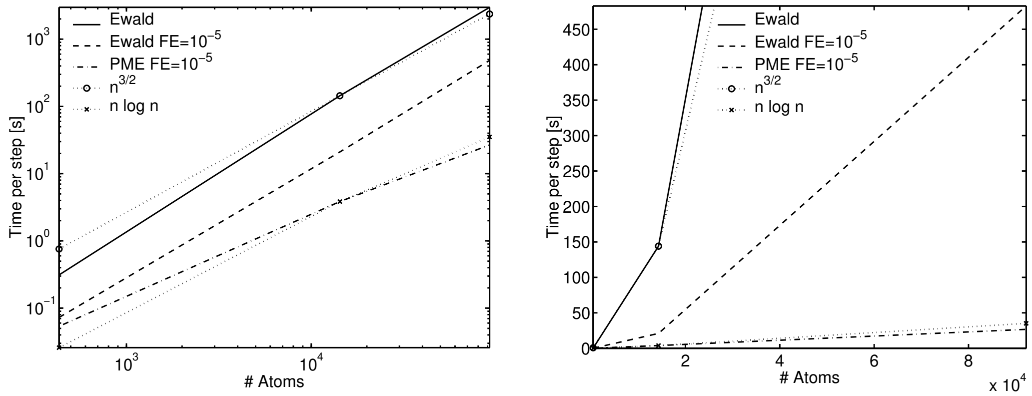


Figure 17: Comparison of fast electrostatic algorithms for periodic boundary conditions. The left figure has logarithmic axes; the right figure has linear axes.

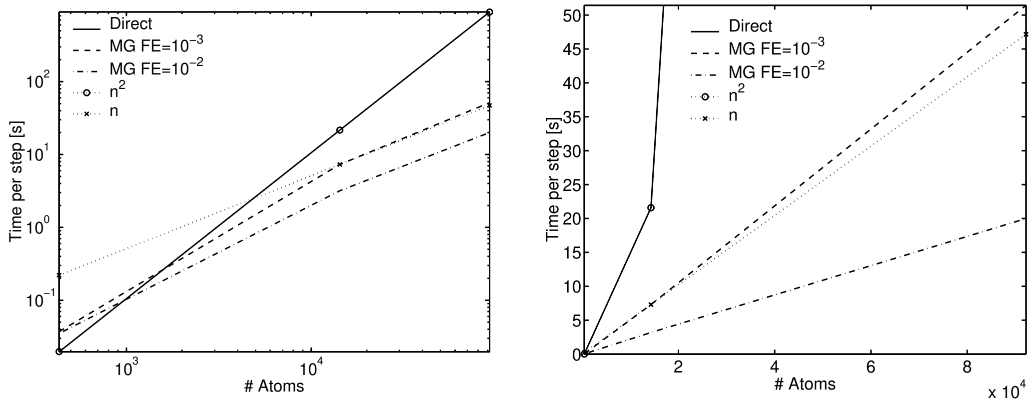


Figure 18: Comparison of fast electrostatic algorithms for vacuum. The left figure has logarithmic axes; the right figure has linear axes.

Test case	# atoms N	Direct [s]	MG 10^{-2} [s]	MG 10^{-3} [s]	Ewald [s]	Ewald 10^{-5} [s]	PME 10^{-5} [s]
Water	432	0.01962	0.03498	0.03707	0.31159	0.07277	0.05365
BPTI	14,281	21.584	3.1818	7.3027	144.035	21.021	3.85
ApoA1	92,224	903.963	19.994	51.441	2978.52	482.847	26.728

Table 4: Performance comparison of fast electrostatic force algorithms on an IBM p690 Regatta Turbo system. The performance numbers represent the average time to evaluate the electrostatic contributions for one time step.

6 Conclusions

The aim of this thesis has been to study three different relevant aspects of molecular dynamics (MD):

- The study of parallelism that is achievable in real-life MD applications and algorithms.

- The design and aspects of a general purpose framework for MD applications.

- The study of the most relevant fast electrostatic force algorithms.

In this thesis, two different parallelization approaches are studied. First, in paper [C] spatial decomposition was evaluated for a particular MD program for 2-dimensional physical problems (see also papers [A] and [B]) with short-range van der Waals interactions. Different partitioning and communication strategies were applied with up to 10^7 particles and 128 processors to demonstrate the parallel scalability. Second, in paper [D] an incremental parallel approach was designed and incorporated into the framework PROTOMOL. Although the representation relies on replicated data, the approach scales well for a moderate number of processors (up to 32). The approach was motivated to provide transparent and encapsulated parallelism, where sequential and parallel implementations can co-exist. It supports the development and integration of *sequential* experimental, and novel algorithms, while concurrently benefiting from already parallelized parts in the framework. During this work, this has emerged as an important feature of the framework, since numerous researchers focus primarily on algorithm development and do not wish to deal with parallelization issues.

The main part of the work is devoted to the design and the development of the fast electrostatic force algorithms. In order to do so, we used a considerable amount of time and effort to extend and enhance the component-based framework PROTOMOL. The front-end was re-designed and the forces were generalized (Policy pattern [42, pp. 315-323]) and extended together with a force factory (Abstract Factory [42, pp. 87-95]). PROTOMOL has been parallelized based on an incremental parallelization approach. The development was realized in collaboration with Jesús A. Izaguirre[20] and his students.

[20] Department of Computer Science and Engineering, University of Notre Dame, USA.

The component-based framework approach gives the needed flexibility to configure and dynamically compose MD applications together with a run-time efficient back-end for the MD computational intensive part.

The PROTOMOL framework is sufficiently general. It has been used in a framework called COMPUCELL [38], which models morphogenesis and other processes of developmental biology at the cellular and organism level. The efficiency of the back-end is demonstrated by a comparison of PROTOMOL against NAMD2 (Sections 5.9.1 and 5.9.2). The dynamic composition is illustrated by the ability to define arbitrary MTS schemes based on an integrator definition language. The framework's flexibility and ease to perform general purpose MD simulations based on the Newton's equation of motion have been demonstrated by a macroscopic simulation of Ugelstad spheres in a magnetic fluid (Appendix D.3). In a laboratory course in computational science at the Department of Informatics at the University of Bergen, students successfully answered the questions of matter by performing simulations with the help of PROTOMOL (Appendix D.2). The framework is also used in an ongoing project to study Coulomb crystals with 10^6 ions (Appendix D.4) in collaboration with experimentalists. Furthermore, the development of sophisticated integrators, such as MOLLY and Dissipative Molly is ongoing work at the University of Notre Dame. Overall, the framework satisfies the requirements of an experimental platform for general MD applications.

The framework has been extended with three fast electrostatic force algorithms: Standard Ewald summation ($\mathcal{O}(N^{\frac{3}{2}})$), smooth particle-mesh based Ewald summation ($\mathcal{O}(N \log N)$) and multi-grid ($\mathcal{O}(N)$), where N is the number of particles/atoms. In Section 5.9.3, the performances of the different methods are compared. Their designs are based on the Policy pattern to provide the flexibility to customize them. All three methods support both vacuum and periodic boundary conditions. In case of multi-grid under periodic boundary conditions, the coarsest grid is evaluated by an ordinary matrix-vector multiplication, which to our knowledge is the best approach to conserve system properties. These fast algorithms also support different splittings for MTS integrator schemes to lengthen the time step and achieve better performance. However, these advanced methods have numerous parameters and it is not always trivial to find the right configuration to achieve optimal run-time efficiency for a given accuracy. Therefore, different instrumental methods are provided to compare run-time efficiency and accuracy, such that the optimal choice of parameters for a given force algorithm can be determined automatically in the future.

The software package PROTOMOL is generally available as a public domain code [63]. It is written in C++ and consists of 40,000 lines of code and 180 C++

classes. The communication and parallelism is based on the Message Passing Interface (MPI). PROTOMOL runs on Solaris, Linux, AIX, and IRIX systems.

Outlook

MD is an active research field, especially the development of the fast electrostatic force algorithms combined with MTS integrator schemes. A major difficulty with fast electrostatic force algorithms is the optimal choice of parameters for a given accuracy and run-time efficiency. It would be desirable to determine these parameters automatically. Furthermore, the choice of the right method is obviously important. There is an ongoing collaboration project at the University of Notre Dame to provide heuristic estimates and algorithms to choose both the right method and optimal parameters.

The fast electrostatic force algorithms should be extended for generic potentials, e.g., van der Waals. This will require a careful extension of the existing design with new policies. For some methods this may however not be trivial.

During this thesis, the requirement to customize the parallelism in more detail was identified as a possible enhancement. This is desirable to be able to select among different communication approaches and strategies of work distribution.

From the framework point of view, there are still several enhancements possible. For example, the integrator design is missing the support of shadow Hamiltonian integrators [106]. These are cheap approximations to the modified Hamiltonian that is close to the Hamiltonian of interest [101]. Furthermore, the integrator design should be revisited once experimental integrators have been proven valuable.

Acknowledgments

I would like to thank my supervisors, Professor Petter E. Bjørstad and co-advisor Professor Jan Petter Hansen[21], for a very challenging and interesting project, and for lots of support. Many thanks go to the Research Council of Norway for funding the project. Also many thanks to assistant professor Jesús A. Izaguirre[22] and his students for their fruitful collaboration and support. Furthermore, I would like to thank the committee, Dr. Godehard Sutmann[23], Professor Hans Petter Langtangen[24] and Professor Tor Sørevik, for the time and effort they have taken to read and comment on this manuscript. Their comments were very useful. I'm also grateful to Jacko Koster for commenting on early drafts of this manuscript. I also owe thanks to the Department of Informatics and Parallab for the excellent working environment. I wish to thank Parallab for giving me unrestricted access to their supercomputer facilities. Last but not least, I would like to thank the "lunch-gang" and KKK.

[21] Department of Physics, University of Bergen.
[22] Department of Computer Science and Engineering, University of Notre Dame, USA.
[23] Central Institute for Applied Mathematics, Research Center Jülich, Jülich, Germany.
[24] Simula Research Laboratory, Oslo.

References

[1] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley, 2001.

[2] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, New York, 1987.

[3] S. Amann, C. Streit, and H. Bieri. BOOGA – A component-oriented framework for computer grahics. In *GraphiCon*, pages 193–200, Moscow, 1997.

[4] N. Attig, M. Lewerenz, G. Sutmann, and R. Vogelsang. DMMD – A modular molecular dynamics program for MPP-systems. 5th SGI/Cray MPP Workshop, Bologna, Italy, `http://www.cineca.it/mpp-workshop`, September 1999.

[5] N. S. Bakhvalov. On the convergence of a relaxation method with natural constraints on the elliptic operator. *USSR Comput. Math. Phys.*, 6(101–135), 1966.

[6] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, 1986.

[7] P. F. Batcho, D. A. Case, and T. Schlick. Optimized particle-mesh Ewald/multiple-time step integration for molecular dynamics simulations. *J. Chem. Phys.*, 115(9):4003–4018, 2001.

[8] P. F. Batcho and T. Schlick. New splitting formulations for lattice summations. *J. Chem. Phys.*, 115(18):8312–8326, November 2001.

[9] D. M. Beazley and P. S. Lomdahl. Message-passing multi-cell molecular dynamics on the Connection Machine 5. *Parallel Computing*, 20:173–195, 1994.

[10] T. Bishop, R. D. Skeel, and K. Schulten. Difficulties with multiple timestepping and the fast multipole algorithm in molecular dynamics. *J. Comp. Chem.*, 18(14):1785–1791, November 15, 1997.

[11] John A. Board, Jr., J. W. Causey, James F. Leathrum, Jr., Andreas Windemuth, and Klaus Schulten. Accelerated molecular dynamics simulation

with the parallel fast multipole algorithm. *Chem. Phys. Lett.*, 198:89–94, 1992.

[12] N. P. Boghossian, O. Kohlbacher, and H.-P. Lenhof. Ball: Biochemical algorithms library. In J. S Vitter and C. D. Zaroliagis, editors, *3rd Int. Workshop on Algorithm Engineering (WAE-99)*, volume 1668 of *In Lecture Notes in Computer Science*, pages 330–344. Springer-Verlag, London, July 1999.

[13] A. Brandt, J. Bernholc, and K. Binder, editors. *Multiscale Computational Methods in Chemistry and Physics*, volume 177 of *NATO Science Series: Series III Computer and Systems Sciences*. IOS Press, Amsterdam, Netherlands, January 2001.

[14] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *J. Comp. Chem.*, 4:187–217, 1983.

[15] B. R. Brooks and M. Hodošček. Parallelization of CHARMM for MIMD machines. *Chemical Design Automation News*, 7:16–22, December 1992.

[16] 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*, 103:170–186, 1997.

[17] B. Bühlmann. *Extensions and Applications of the Graphics Framework BOOGA*. PhD thesis, Institute of Computer Science and Applied Mathematics, University of Berne, 1998.

[18] D. Bulka and D. Mayhew. *Efficient C++, Performance Programming Techniques*. Addison-Wesley, 1999.

[19] *Common Component Architecture Forum*. See `http://www.acl.lanl.gov/cca-forum`.

[20] D. Chappell. *Understanding ActiveX and OLE*. Microsoft Press, 1997.

[21] Z. M. Chen, T. Cagin, and W. A. Goddard. Fast Ewald sums for general van der Waals potentials. *J. Comp. Chem.*, 18:1365–1370, 1997.

[22] T. W. Clark, R. van Hanxleden, J. A. McCammon, and L. Ridgway. Parallelization strategies for a molecular dynamics program. In *Intel Technology Focys Conference Oroceedings*, April 1992.

[23] T. W. Clark, R. van Hanxleden, J. A. McCammon, and L. R. Scott. Parallelizing molecular dynamics using spatial decomposition. In *Proceedings of the Scalable High-Performance Computing Conference*, pages 95–102, Los Alamitos, California, 1994. IEEE Computer Society Press.

[24] *The Common Object Request Broker: Architecture and Specification*, December 2001. Revision 2.6.

[25] B. Coulange. *Software Reuse*. Springer Verlag, London, 1998.

[26] T. A. Darden, D. M. York, and L. G. Pedersen. Particle mesh Ewald. An $N \cdot \log(N)$ method for Ewald sums in large systems. *J. Chem. Phys.*, 98:10089–10092, 1993.

[27] M. Deserno and C. Holm. How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines. *J. Chem. Phys.*, 109(18):7678–7693, 1998.

[28] H.-Q. Ding, N. Karasawa, and W. A. Goddard III. The reduced cell multipole method for coulomb interactions in periodic systems with million-atom unit cell. *Chem. Phys. Lett.*, 196(6):6–10, August 1992.

[29] Z. H. Duan and R. Krasny. An Ewald summation based multipole method. *J. Chem. Phys.*, 113(9):3492–3495, 2000.

[30] M. J. Dudek and J. W. Ponder. Accurate modeling of the intramolecular electrostatic energy of proteins. *J. Comp. Chem.*, 16:791–816, 1995.

[31] M. Eichinger, H. Grubmüller, H. Heller, and P. Tavan. FAMUSAMM: An algorithm for rapid evaluation of electrostatic interactions in molecular dynamics simulations. *J. Comp. Chem.*, 18:1729–1749, 1997.

[32] R. Esser, P. Grassberger, J. Grotendorst, and M. Lewerenz. EGO – an efficient molecular dynamics program and its application to protein dynamics simulations. In *Workshop on Molecular Dynamics on Parallel Computers*, World Scientific, pages 154–174, Singapore 912805, 2000.

[33] U. Essmann, L. Perera, and M. L. Berkowitz. A smooth particle mesh Ewald method. *J. Chem. Phys.*, 103(19):8577–8593, 1995.

[34] P. Ewald. Die Berechnung optischer und elektrostatischer Gitterpotentiale. *Ann. Phys.*, 64:253–287, 1921.

[35] R. P. Fedorenko. A relaxation method for solving elliptic differential equations. *USSR Comput. Math. Phys.*, 1:1092, 1961.

[36] D. Fincham. Optimisation of the Ewald sum for large systems. *Mol. Sim.*, 13:1–9, 1994.

[37] M. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, C(21):94, 1972.

[38] Laboratory for Computational Life Sciences. `http://www.nd.edu/~lcls/compucell`, 2002. CompuCell home page.

[39] T. Forester and W. Smith. On multiple time-step algorithms and the Ewald sum. *Mol. Sim.*, 13(3):195–204, 1994.

[40] MPI Forum. MPI-2: Extensions to the Message-Passing Interface. `http://www.mpi-forum.org/docs/docs.html`.

[41] MPI Forum. MPI: Message-Passing Interface. `http://www.mpi-forum.org/docs/docs.html`.

[42] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison-Wesley, Reading, Massachusetts, 1995.

[43] G. Gao. *Large Scale Molecular Simulations with Application to Polymers and Nano-scale Materials*. PhD thesis, Caltech, 1998.

[44] B. García-Archilla, J. M. Sanz-Serna, and R. D. Skeel. Long-time-step methods for oscillatory differential equations. *SIAM J. Sci. Comput.*, 20(3):930–963, October 20, 1998.

[45] B. García-Archilla, J. M. Sanz-Serna, and R. D. Skeel. The mollified impulse method for oscillatory differential equations. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis 1997*, pages 111–123, London, 1998. Pitman.

[46] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. PVM 3 users guide and reference manual. Technical Manual ORNL/TM-12187, Oak Ridge National Laboratory, May 1994.

[47] H. Goldstein. *Classical mechanics*. Addison-Wesley, Reading, Massachusetts, 1980.

[48] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73:325–348, 1987.

[49] H. Grubmüller, H. Heller, A. Windemuth, and K. Schulten. Generalized Verlet algorithm for efficient molecular dynamics simulations with long-range interactions. *Molecular Simulation*, 6:121–142, 1991.

[50] B. Hafskjold and L. Rekvig. Comparison between PROTOMOL and NEMD for a simple Lennard-Jones system. Technical report, Department of Chemistry, Trondheim, NTNU, March 2002.

[51] D. Hardy. *Molecular Dynamics API*. Theoretical Biophysics Group, University of Illinois at Urbana-Champaign, 405 North Matthews Avenue, Urbana, IL 61801, 0.96 edition, October 1999. Download at `http://www.ks.uiuc.edu/~dhardy/mdapi`.

[52] R. H. Hasse and V. V. Avilov. Structure and Mandelung energy of spherical Coulomb crystals. *Phys. Rev. A*, 44(7):4506–4515, 1991.

[53] J. Hermans, R. H. Yun, J. Leech, and D. Cavanaugh. SIGMA: SI-mulations of MA-cromolecule. `http://femto.med.unc.edu/SIGMA`.

[54] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. McGraw-Hill, New York, 1981.

[55] Y.-S. Hwang, R. Das, J. H. Saltz, M. Hodošček, and B. R. Brooks. Parallelizing molecular dynamics programs for distributed-memory machines. *IEEE Computational Science & Engineering*, 2(2):18–29, Summer 1995.

[56] B. Isralewitz, M. Gao, and K. Schulten. Steered molecular dynamics and mechanical functions of proteins. *Curr. Opinion Struct. Biol.*, 2001. Invited Article.

[57] J. A. Izaguirre. *Longer Time Steps for Molecular Dynamics*. PhD thesis, University of Illinois at Urbana-Champaign, 1999. Also UIUC Technical Report UIUCDCS-R-99-2107.

[58] J. A. Izaguirre. Generalized mollified multiple time stepping methods for molecular dynamics. In A. Brandt, J. Bernholc, and K. Binder, editors, *Multiscale Computational Methods in Chemistry and Physics*, volume 177 of *NATO Science Series: Series III Computer and Systems Sciences*, pages 34–47. IOS Press, Amsterdam, Netherlands, Jan 2001.

[59] J. A. Izaguirre, D. P. Catarello, J. M. Wozniak, and R. D. Skeel. Langevin stabilization of molecular dynamics. *J. Chem. Phys.*, 114(5):2090–2098, February 1, 2001.

[60] J. A. Izaguirre, Q. Ma, T. Matthey, J. Willcock, T. Slabach, B. Moore, and G. Viamontes. Overcoming instabilities in Verlet-I/r-RESPA with the mollified impulse method. In T. Schlick and H. H. Gan, editors, *Proceedings of 3rd International Workshop on Methods for Macromolecular Modeling*, volume 24 of *Lecture Notes in Computational Science and Engineering*, pages 146–174. Springer-Verlag, Berlin, New York, 2002.

[61] J. A. Izaguirre, T. Matthey, J. Willcock, Q. Ma, B. Moore, T. Slabach, and G. Viamontes. A tutorial on the prototyping of multiple time stepping integrators for molecular dynamics. Available from `http://www.cse.nd.edu/~lcls/Protomol.html`, 2001.

[62] J. A. Izaguirre, S. Reich, and R. D. Skeel. Longer time steps for molecular dynamics. *J. Chem. Phys.*, 110(19):9853–9864, May 15, 1999.

[63] J. A. Izaguirre, J. Willcock, T. Matthey, Q. Ma, T. B. Slabach, T. Steinbach, S. Stender, G. F. Viamontes, and J. Mohnke. PROTOMOL: An object oriented framework for molecular dynamics. Available online via `http://www.cse.nd.edu/~lcls/Protomol.html`, 2000–2002.

[64] R. E. Johnson. Frameworks. Seminar, Zühlke Informatik, Zürich, May 1996.

[65] R. E. Johnson and B. Foote. Designing reusable classes. *J. of Object-Oriented Programming*, June/July 1988.

[66] L. Kalé, R. Skeel, R. Brunner, M. Bhandarkar, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *J. Comput. Phys*, 151(1):283–312, May 1, 1999.

[67] N. Karawasa and W. A. Goddard. Acceleration of convergence for lattice sums. *J. Phys. Chem.*, 93:7320, 1989.

[68] M. Kawata and M. Mikami. Acceleration of the canonical molecular dynamics simulation by the particle mesh Ewald method combined with the multiple time-step integrator algorithm. *Chemical Physics Letters*, 313(1-2):261–266, 1999.

[69] M. Kawata and M. Mikami. Rapid calculation of two-dimensional Ewald summation. *Chemical Physics Letters*, 340(1-2):157–164, 2001.

[70] M. Kawata and U. Nagashima. Particle mesh Ewald method for three-dimensional systems with two-dimensional periodicity. *Chemical Physics Letters*, 340(1-2):165–172, 2001.

[71] J. Kolafa and J. W. Perram. Cutoff errors in the Ewald summation formulae for point charge systems. *Molecular Simulation*, 9:351–368, 1992.

[72] Y. Komeiji. Ewald summation and multiple time step methods for molecular dynamics simulation of biological molecules. *Journal of Molecular Structure (THEOCHEM)*, 530(3):237–243, 2000.

[73] C. G. Lambert, T. A. Darden, and J. A. Board. A multipole-based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles. *J. Comp. Phys.*, 126(2):274–287, July 1996.

[74] D. J. Langridge, J. F. Hart, and S. Crampin. Ewald summation technique for one-dimensional charge distributions. *Computer Physics Communications*, 134(1):78–85, 2001.

[75] Andrew R. Leach. *Molecular Modelling: Principles and Applications*. Addison-Wesley Longman, Reading, Massachusetts, July 1996.

[76] S. W. De Leeuw, J. W. Perram, and E. R. Smith. Simulation of electrostatic systems in periodic boundary conditions. I. lattice sums and dielectric constants. *Proc. R. Soc. Lond.*, A 373:27–56, 1980.

[77] E. Lindahl, B. Hess, and D. van der Spoel. GROMACS 3.0: A package for molecular simulation and trajectory. *JMM*, 7(8):306–317, 2001.

[78] T. Lippert, A. Seyfried, A. Bode, and K. Schilling. Hyper-systolic parallel computing. *IEEE Trans. Paral. Distr. Syst.*, 9:97–108, 1998.

[79] T. R. Littell, R. D. Skeel, and M. Zhang. Error analysis of symplectic multiple time stepping. *SIAM J. Numer. Anal.*, 34(5):1792–1807, October 1997.

[80] P. S. Lomdahl, P. Tamayo, N. Grønbech-Jensen, and D. M. Beazley. 50 G-Flops molecular dynamics on the Connection Machine. In *Proceedings of Supercomputing '93*, pages 520–527. IEEE Computer Society, 1993.

[81] A. P. Lyubartsev and A. Laaksonen. MDynaMix – a scalable portable parallel MD simulation package for arbitrary molecular mixtures. *Comput. Phys. Commun.*, 128:565–589, 2000.

[82] Q. Ma and J. A. Izaguirre. Targeted mollified impulse method for molecular dynamics. Submitted to *J. Chem. Phys.*, 2002.

[83] T. Matthey. Objektorientierte Gebäudemodellierung. Master's thesis, Institute of Computer Science and Applied Mathematics, University of Berne, 1997.

[84] E. Merzbacher. *Quantum Mechanics*. John Wiley & Sons, 3rd edition edition, 1998.

[85] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, 2nd edition, 1997.

[86] M. Nelson, W. Humphrey, A. Gursoy, A. Dalke, L. Kalé, R. D. Skeel, and K. Schulten. NAMD – A parallel, object-oriented molecular dynamics program. *Int. J. Supercomput. Applics. High Performance Computing*, 10(4):251–268, Winter 1996.

[87] D. Okunbor and R. Murty. Parallel molecular dynamics using force decomposition. In P. Deuflhard, J. Hermans, B. Leimkuhler, A. Mark, S. Reich, and R. D. Skeel, editors, *Computational Molecular Dynamics: Challenges, Methods, Ideas*, volume 4 of *Lecture Notes in Computational Science and Engineering*, pages 483–494. Springer-Verlag, November 1998.

[88] OpenMP-forum. OpenMP: A proposed industry standard API for shared memory programming. technical report. `http://www.openmp.org`, October 1997.

[89] J. W. Perram, H. G. Petersen, and S. W. de Leeuw. An algorithm for the simulation of condensed matter which grows as the $\frac{3}{2}$ power of the number of particles. *Mol. Phys.*, 65:875–893, 1988.

[90] H. G. Petersen. Accuracy and efficiency of the particle mesh Ewald method. *J. Chem. Phys.*, 103(3668-3679), 1995.

[91] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Chem. Phys.*, 117:1–19, 1995.

[92] S. Plimpton and B. Hendrickson. A new parallel method for molecular dynamics simulation of macromolecular systems. *J. Comp. Chem.*, 17(3):326, 1996.

[93] E. L. Pollock and J. Glosli. Comments on PPPM, FMM, and the Ewald method for large periodic Coulombic systems. *Computer Physics Communications*, 95:93–110, 1996.

[94] L. Hornekær. *Single- and Multi-Species Coulomb Ion Crystals: Structures, Dynamics and Sympathetic Cooling*. PhD thesis, University of Aarhus, Denmark, November 2000.

[95] W. Rankin and J. Board. A portable distributed implementation of the parallel multipole tree algorithm. *IEEE Symposium on High Performance Distributed Computing*, 1995. [Duke University Technical Report 95-002].

[96] W. T. Rankin. *Efficient Parallel Implementations of Multipole Based N-Body Algorithms*. PhD thesis, Duke University, 1997.

[97] D. C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 1995.

[98] K. Refson. Moldy: A portable molecular dynamics simulation program for serial and parallel computers. *Comput. Phys. Commun.*, 126(3):309–328, 2000.

[99] J. Roth, F. Gähler, and H.-R. Trebin. A molecular dynamics run with 5.180.116.000 particles. *Int. J. of Modern Phys. C*, 11(2):317–322, 2000.

[100] C. Sagui and T. Darden. Multigrid methods for classical molecular dynamics simulations of biomolecules. *J. Chem. Phys.*, 114(15):6578–6591, 2001.

[101] J. M. Sanz-Serna and M. P. Calvo. *Numerical Hamiltonian Problems*. Chapman and Hall, London, 1994.

[102] J. P. Schiffer. Phase transitions in anisotropically confined ionic crystals. *Physical Review Letters*, 70(6):818–821, 1993.

[103] SGI. The Standard Template Library: Introduction. `http://www.sgi.com/tech/stl`.

[104] J. Shimada, H. Kaneko, and T. Takada. Performance of fast multipole methods for calculating electrostatic interactions in biomacromolecular simulations. *J. Comp. Chem.*, 15(1):28, 1994.

[105] J. G. Siek and A. Lumsdaine. The Matrix Template Library: A unifying framework for numerical linear algebra. In *International Symposium on Computing in Object-Oriented Parallel*, Parallel Object Oriented Scientific Computing. ECOOP, 1998.

[106] R. D. Skeel and D. J. Hardy. Practical construction of modified Hamiltonians. *SIAM J. on Sci. Comput.*, 23(4):1172–1188, November 2001.

[107] R. D. Skeel, I. Tezcan, and D. J. Hardy. Multiple grid methods for classical molecular dynamics. *J. Comp. Chem.*, 23(6):673–684, March 2002.

[108] G. D. Smith, C. Ayyagari, and D. Bedrov. Lucretius: A molecular dynamics simulation program, 1999. `http://www.che.utah.edu/~gdsmith/mdcode/main.html`.

[109] L. Smith. Molecular science modelling. Technical report, Edinburgh Parallel Computing Center, The University of Edinburgh, UK, 1997.

[110] W. Smith and T. R. Forester. DL_POLY_2.0: A general-purpose parallel molecular dynamics simulation package. *J. Mol. Graphics*, 14(3), June 1996.

[111] J. Stone, J. Gullingsrud, and K. Schulten. A system for interactive molecular dynamics. In *2001 Symposium on Interactive 3D Graphics*. ACM, 2001. In Press.

[112] C. Streit. *BOOGA, ein Komponentenframework für Grafikanwendungen*. PhD thesis, Institute of Computer Science and Applied Mathematics, University of Berne, 1997.

[113] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, third edition, 1997.

[114] G. Sutmann. Classical dynamics. In J. Grotendorst, D. Marx, and A. Muramatsu, editors, *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms*, volume 10 of *NIC*, pages 211–254, Forschungszentrum Jülich, Germany, 2000. John von Neumann Institute for Computing, NIC-Directors.

[115] H. Sutter. *Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions*. Addison-Wesley, 2000.

[116] H. Tosuji, T. Kishimoto, C. Totsuji, and K. Tsuruta. Competition between two forms of ordering in finite Coulomb clusters. *Physical Review Letters*, 88(12), March 2002.

[117] A. Y. Toukmaji and J. A. Board Jr. Ewald summation techniques in perspective: A survey. *Computer Physics Communications*, 95:73–92, 1996.

[118] M. Tuckerman, B. J. Berne, and G. J. Martyna. Reversible multiple time scale molecular dynamics. *J. Chem. Phys.*, 97(3):1990–2001, 1992.

[119] W. F. van Gunsteren and H. J. C. Berendsen. Computer simulation of molecular dynamics: Methodology, applications and perspectives in chemistry. *Chemie Int. Ed. Engl.*, 29:922–1023, 1990.

[120] T. Veldhuizen. Blitz++: The library that thinks it is a compiler. Conference presentation, Extreme! Computing Laboratory, Indiana University Computer Science Department, Sep. 1998.

[121] J. Vincent and K. M. Merz. A highly portable parallel implementation of AMBER using the Message Massing Interface standard. *J. Comp. Chem.*, 11:1420–1427, 1995.

[122] Z. Wang and C. Holm. Estimate of the cutoff errors in the Ewald summation for dipolar systems. *J. Chem. Phys.*, 115(14):6351–6359, 2001.

[123] Z. Wang, J. Lupo, A. McKenney, and R. Pachter. Large scale molecular dynamics simulations with fast multipole implementations. In *Proceedings of Supercomputing '99*, Portland, Oregon, November 1999.

[124] P. K. Weiner and P. A. Kollman. AMBER: Assisted model building with energy refinement. a general program for modeling molecules and their interactions. *J. Comp. Chem.*, 2:287, 1981.

[125] A. Windemuth. Advanced algorithms for molecular dynamics simulation: The program PMD. In Timothy G. Mattson, editor, *Parallel Computing in Computational Chemistry*, Washington, D.C., 1995. ACS Books. In press.

[126] D. York and W. T. Yang. The fast Fourier–Poisson method for calculating Ewald sums. *J. Chem. Phys.*, 101:3298–3300, 1994.

[127] M. Q. Zhang and R. D. Skeel. Symplectic integrators and the conservation of angular momentum. *J. Comput. Chem.*, 16:365–369, March 1995.

[128] R. Zhou and B. J. Berne. A new molecular dynamics method combining the reference system propagator algorithm with a fast multipole method for simulating proteins and other complex systems. *J. Chem. Phys.*, 103(21):9444–9459, 1995.

[129] R. Zhou, E. Harder, H. Xu, and B. J. Berne. Efficient multiple time step method for use with Ewald and particle mesh Ewald for large biomolecular systems. *J. Chem. Phys.*, 115(5):2348–2358, August 1 2001.

A Publications

B Design

R2: Boundary conditions (TBC)	**VacuumBoundaryConditions** **PeriodicBoundaryConditions**
R3: Cell manager (CM)	**CubicCellManager**
On atom pair (TOneAtomPair)	**OneAtomPair**`<TBC, TCM, TSF, TNF, useSF>` **OneAtomPairTwo**`<TBC, TCM, TSF1, TNF1, TSF2, TNF2, useSF>` **OneAtomPairMollifyLJ**`<TBC, TCM, TSF, TNF, useSF>` **OneAtomPairMollifyTwo**`<TBC, TCM, TSF1, TNF1, TSF2, TNF2, useSF>`
R4: Non-bonded force	**CoulombForce** **GravitationForce** **LennardJonesForce** **MagneticDipoleForce**
R5: Switching function (TSF)	**C1SwitchingFunction** **C2SwitchingFunction** **ComplementSwitchingFunction**`<TSF>` **CutoffSwitchingFunction** **RangeSwitchingFunction**`<TSF>` **ShiftSwitchingFunction** **UniversalSwitchingFunction**

Table 5: Overview of policy choices for the non-bonded force algorithms (**R1**): cutoff (or truncation), simple-full (or direct) and full (encounters also pairs over different unit MD cells). **OneAtomPair** computes one interaction pair, where **OneAtomPairTwo** computes two potentials simultaneously. **OneAtomPairMollifyLJ** and **OneAtomPairMollifyTwo** are the corresponding interactions for MOLLY integrators.

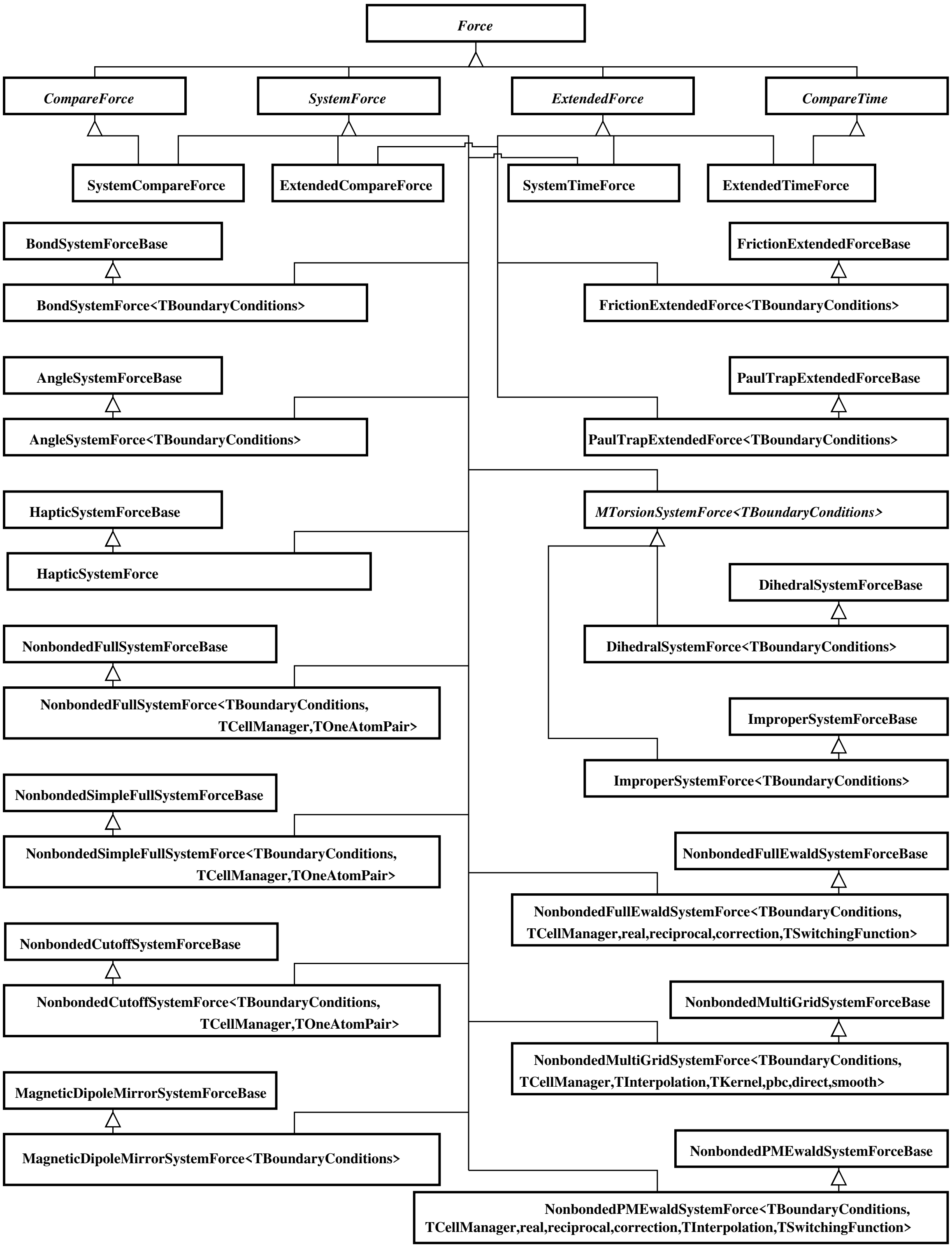


Figure 19: Overview design of forces.

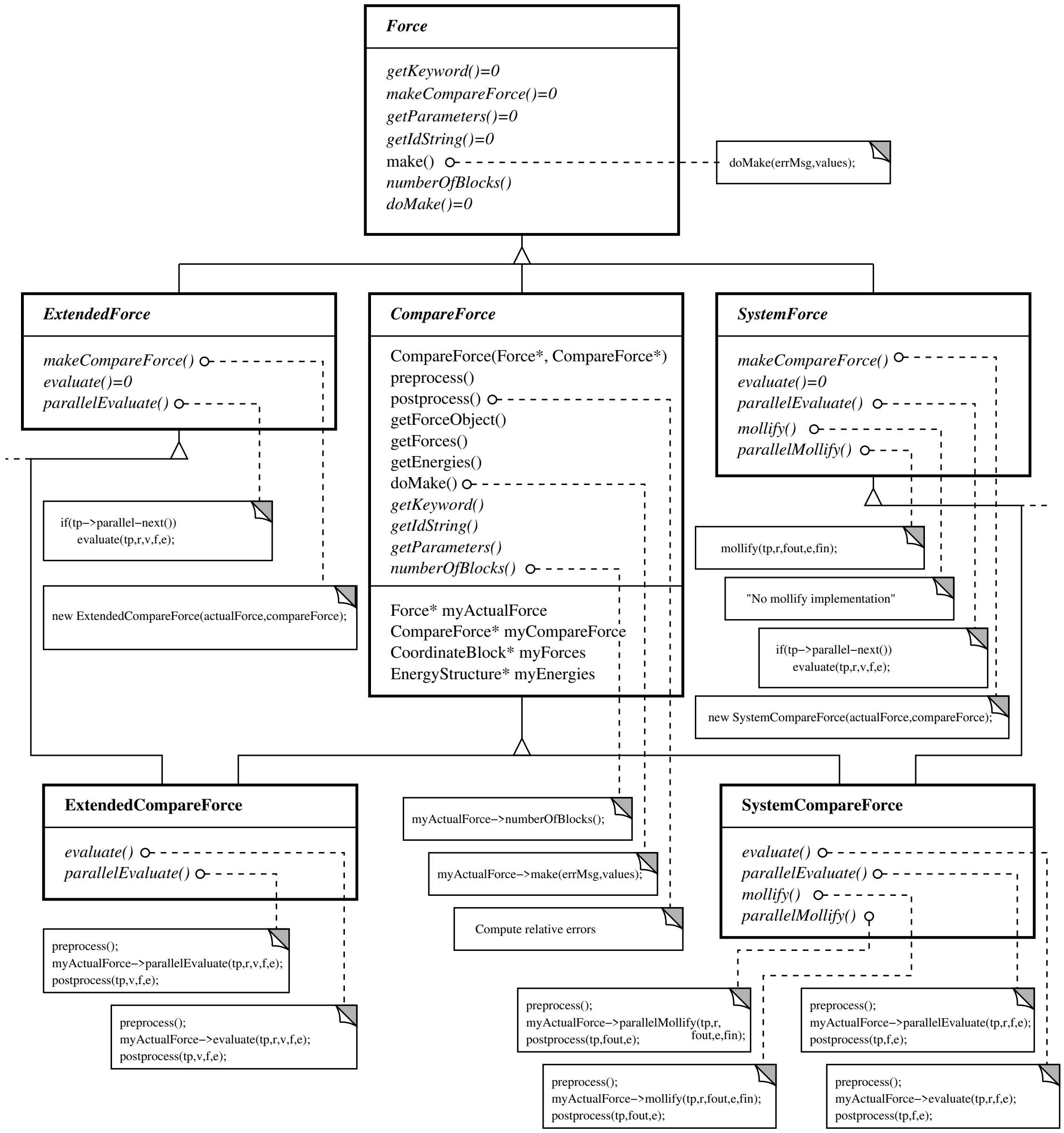


Figure 20: Top level design of forces.

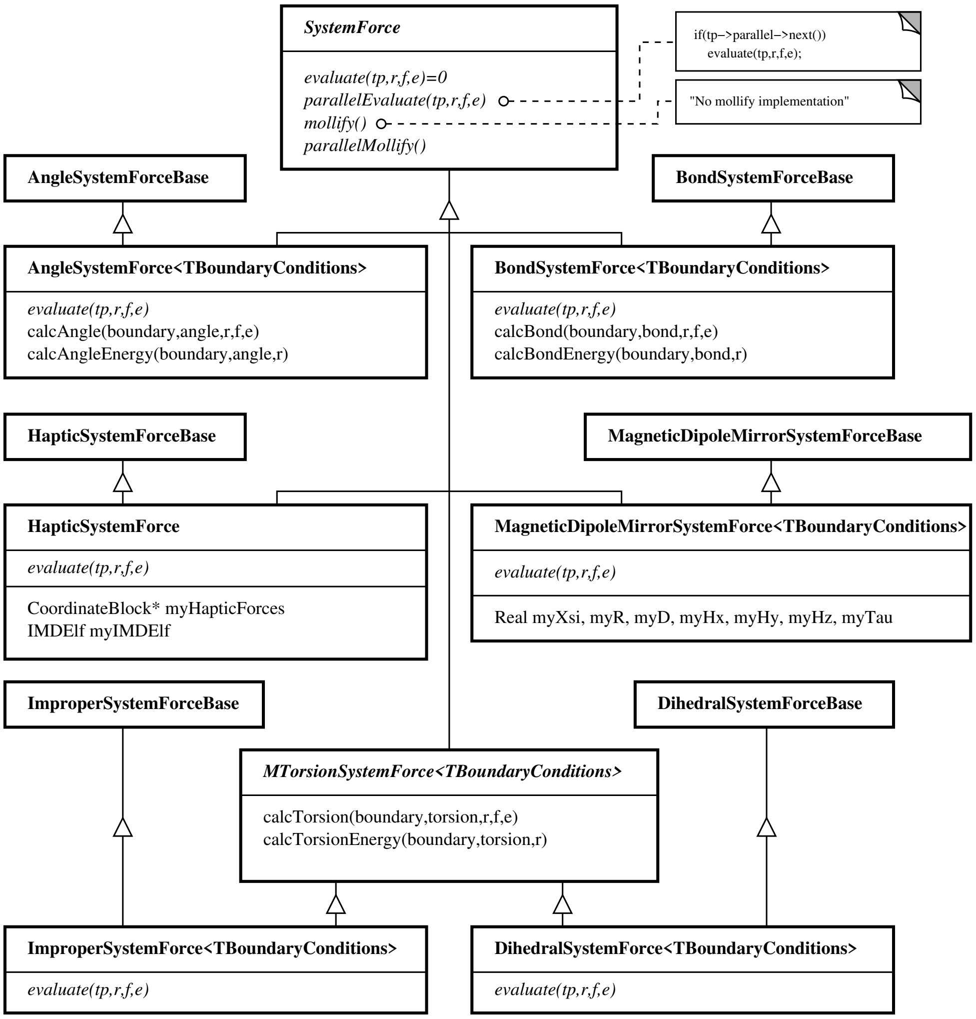


Figure 21: Detailed design of bonded system forces.

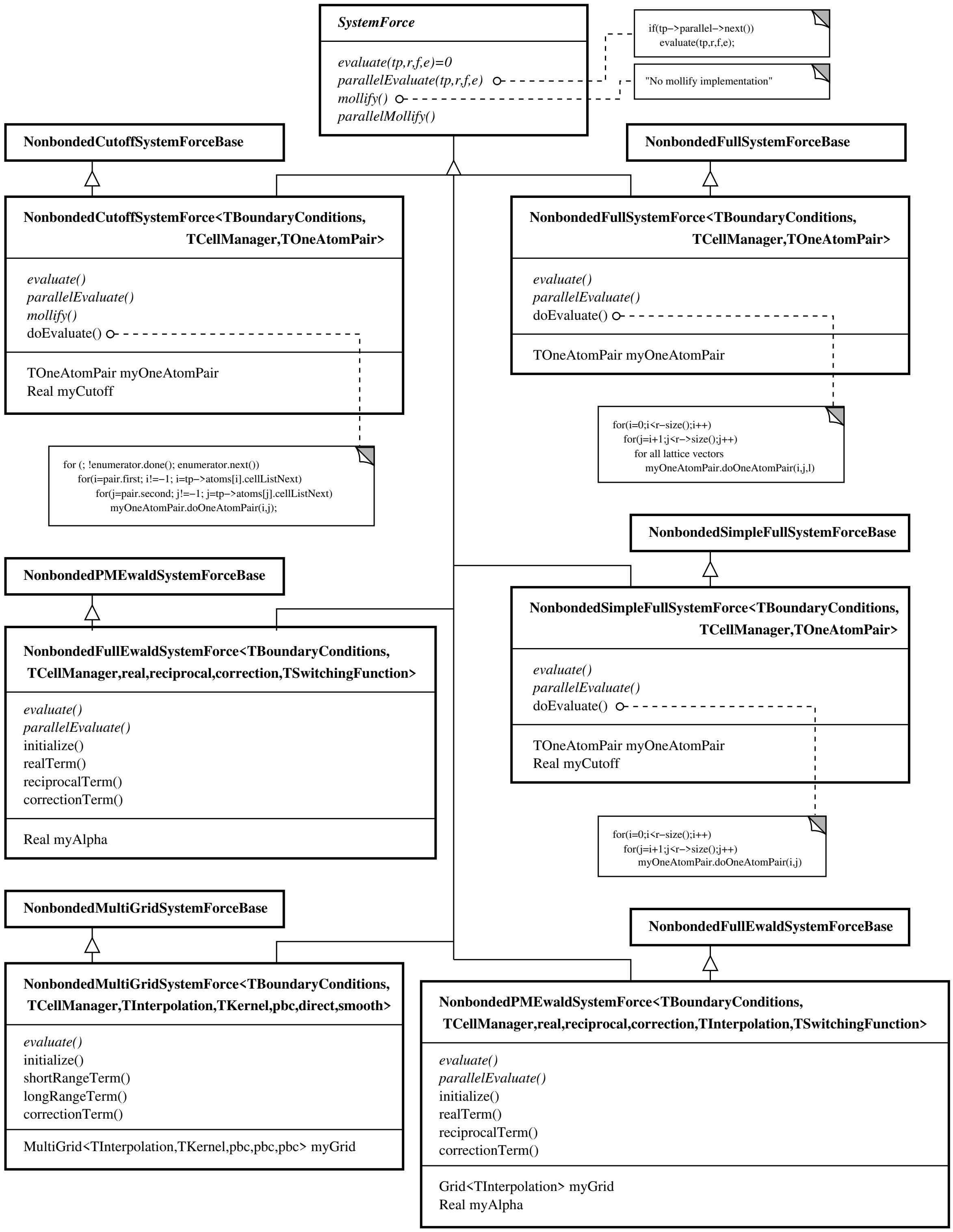


Figure 22: Detailed design of non-bonded system forces.

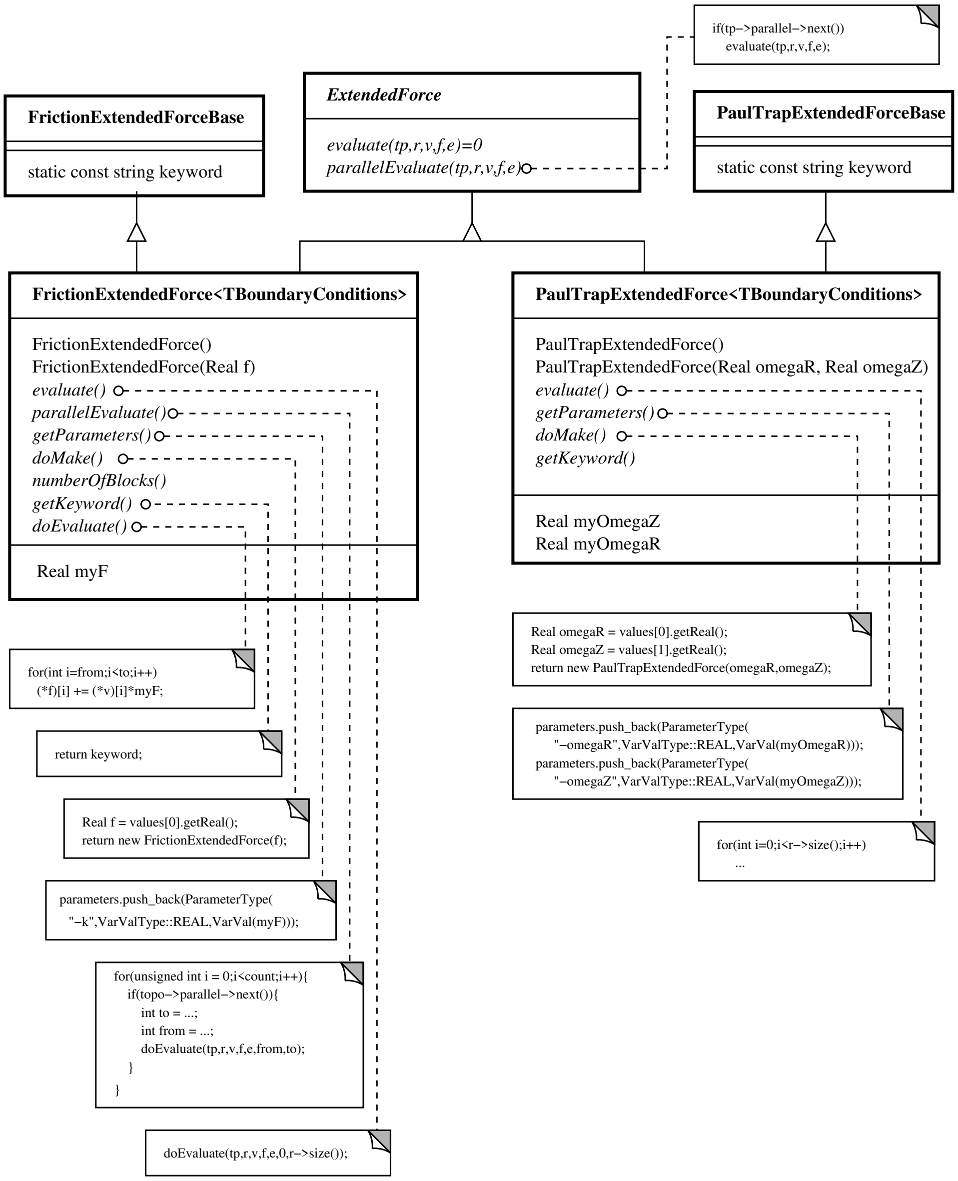


Figure 23: Detailed design of extended forces.

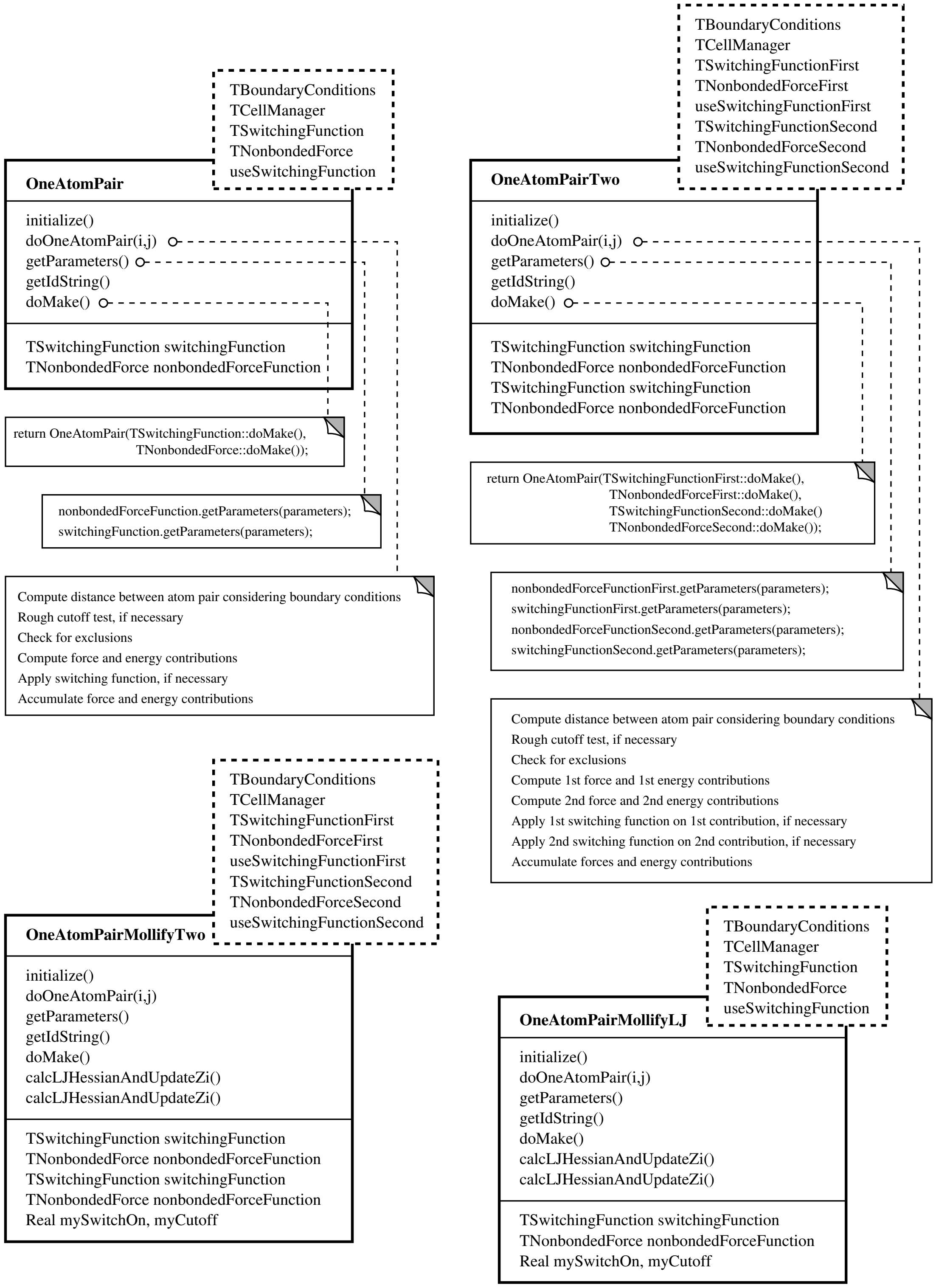


Figure 24: Detailed design of `TOneAtomPair`, evaluating one interaction pair or two simultaneously.

C Code

```
#include "ExtendedForce.h"
#include "PaulTrapExtendedForceBase.h"

template<class TBoundaryConditions>
class PaulTrapExtendedForce : public ExtendedForce,
                              private PaulTrapExtendedForceBase {
public:
  PaulTrapExtendedForce():myOmegaXY(0.0),myOmegaZ(0.0){};
  PaulTrapExtendedForce(Real omeagXY, Real omegaZ):myOmegaXY(omegaXY),myOmegaZ(omegaZ){};

public: // From class SystemForce
  virtual void evaluate(const GenericTopology* topo,
                        const CoordinateBlock* positions,
                        const CoordinateBlock* velocities,
                        CoordinateBlock* forces,
                        EnergyStructure* energies){
    const TBoundaryConditions &boundary =
      (dynamic_cast<const SemiGenericTopology<TBoundaryConditions>& >(*topo)).boundaryConditions;
    Real e = 0.0;
    for(int i=0;i<topo->atoms.size();i++){
      Real c = topo->atomTypes[topo->atoms[i].type].mass;
      Coordinates pos(boundary.basisPosition((*positions)[i]));
      Coordinates f(c*myOmegaXY*myOmegaXY*pos.x,
                    c*myOmegaXY*myOmegaXY*pos.y,
                    c*myOmegaZ*myOmegaZ*pos.z);
      (*forces)[i] -= f;
      e += 0.5*c*(myOmegaXY*myOmegaXY*(pos.x*pos.x+pos.y*pos.y)+
                  myOmegaZ*myOmegaZ*pos.z*pos.z);
    }
    energies->otherEnergy += e;
  }

public: // From class Force
  virtual string getKeyword() const{return keyword;}
  virtual string getIdString() const{return keyword;}
  virtual void getParameters(vector<ParameterType>& parameters) const{
    parameters.push_back(ParameterType("-omegaXY",VarValType::REAL,
                                       VarVal(myOmegaXY)));
    parameters.push_back(ParameterType("-omegaZ",VarValType::REAL,
                                       VarVal(myOmegaZ)));
  }
  virtual unsigned int getParametersCount() const{return 2;}
protected:
  virtual Force* doMake(string&, const vector<VarVal>&) const{
    Real omegaXY = values[0].getReal();
    Real omegaZ = values[1].getReal();
    return new PaulTrapExtendedForce(omegaXY,omegaZ);
  }

private: // My data members
  Real myOmegaXY;
  Real myOmegaZ;
};
```

Program 13: Detailed implementation of a Paul Trap attraction.

```
#include "ExtendedForce.h"
#include "FrictionExtendedForceBase.h"

template<class TBoundaryConditions>
class FrictionExtendedForce : public ExtendedForce, private FrictionExtendedForceBase {

public:
  FrictionExtendedForce():myF(0.0){};
  FrictionExtendedForce(Real f):myF(f){};
private:
  void doEvaluate(const GenericTopology* topo, const CoordinateBlock* positions,
                  const CoordinateBlock* velocities, CoordinateBlock* forces,
                  EnergyStructure* energies, int from, int to){
    for(int i=from;i<to;i++)
      (*forces)[i] += (*velocities)[i]*myF;
  }

public: // From class SystemForce
  virtual void evaluate(const GenericTopology* topo, const CoordinateBlock* positions,
                        const CoordinateBlock* velocities, CoordinateBlock* forces,
                        EnergyStructure* energies){
    doEvaluate(topo,positions,velocities,forces,energies,0,positions->size());
  }
virtual void parallelEvaluate(const GenericTopology* topo, const CoordinateBlock* positions,
                              const CoordinateBlock* velocities, CoordinateBlock* forces,
                              EnergyStructure* energies){
    int n = positions->size();
    int num = topo->parallel->getAvailableNum();
    int count = 0;
    if(num >= n) count = n;
    else if(num*2 <= n) count = 2*num;
    else count = num;

    for(int i = 0;i<count;i++){
      if(topo->parallel->next()){
      int to = (n*(i+1))/count;
        if(to > (int)n) to = n;
        int from = (n*i)/count;
        doEvaluate(topo,positions,velocities,forces,energies,from,to);
  } } }

public: // From class Force
  virtual string getKeyword() const{return keyword;}
  virtual string getIdString() const{return keyword;}
  virtual void getParameters(vector<ParameterType>& parameters) const{
    parameters.push_back(ParameterType("-k",VarValType::REAL,VarVal(myF)));
  }
  virtual unsigned int getParametersCount() const{return 1;}
  virtual void numberOfBlocks(const GenericTopology* topo,
                              const CoordinateBlock* pos,
                              unsigned int& n,unsigned  int& step){
    int count = pos->size();
    int num = (unsigned  int)topo->parallel->getAvailableNum();
    if(num >= count)
      n = count;
    else if(num*2 <= count)
      n = 2*num;
    else
      n = num;
    step = 1;
  }
protected:
  virtual Force* doMake(string&, const vector<VarVal>&) const{
    Real f = values[0].getReal();
    return new FrictionExtendedForce(f);
  }

private: // My data members
  Real myF;
};
```

Program 14: Detailed implementation of a velocity dependent friction.

D Gallery

D.1 Biomolecules

The following pictures were generated with PROTOMOL and raytraced[25] with the graphics framework BOOGA [3, 17, 112].

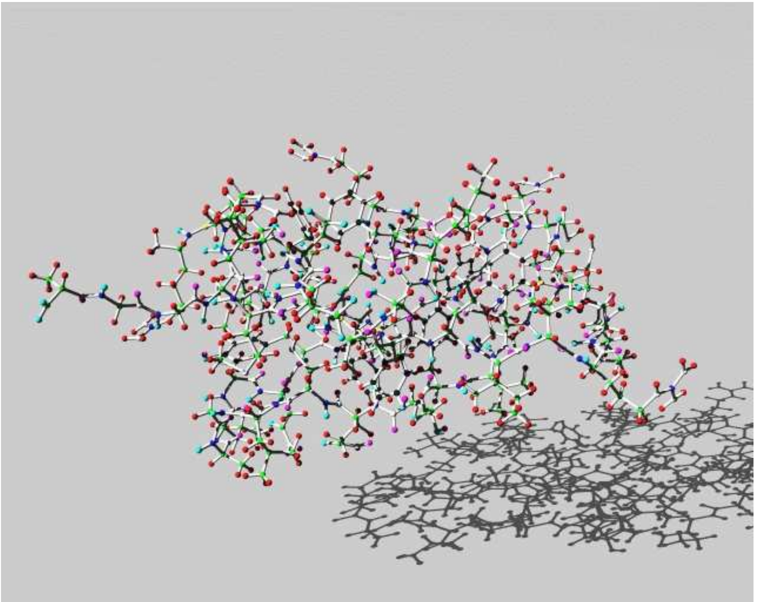


Figure 25: Bovine pancreatic trypsin inhibitor (BPTI) without water, 898 atoms; a 58 amino acid polypeptide that adopts a tertiary fold comprising two strands of antiparallel beta sheet and two short segments of alpha helix.

[25] The raytrace method is a well-known method in computer graphics to generate realistic pictures with shadow, reflection and textures.

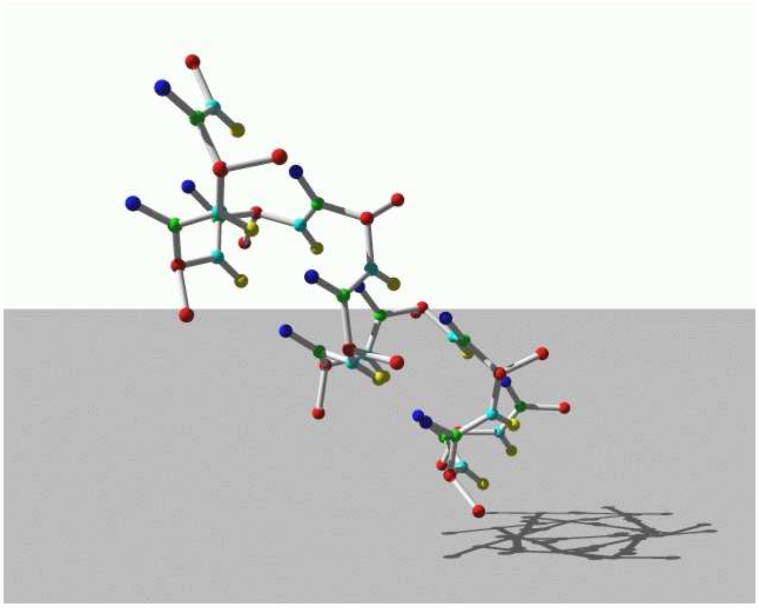


Figure 26: Alanin, 66 atoms. A nonessential amino acid, alpha-aminopropanoic acid, occurring in proteins.

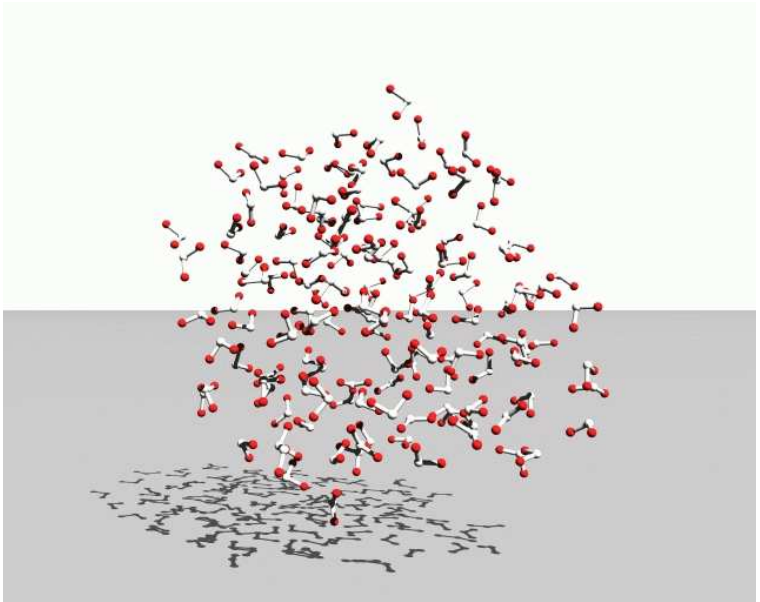


Figure 27: 141-molecule water system.

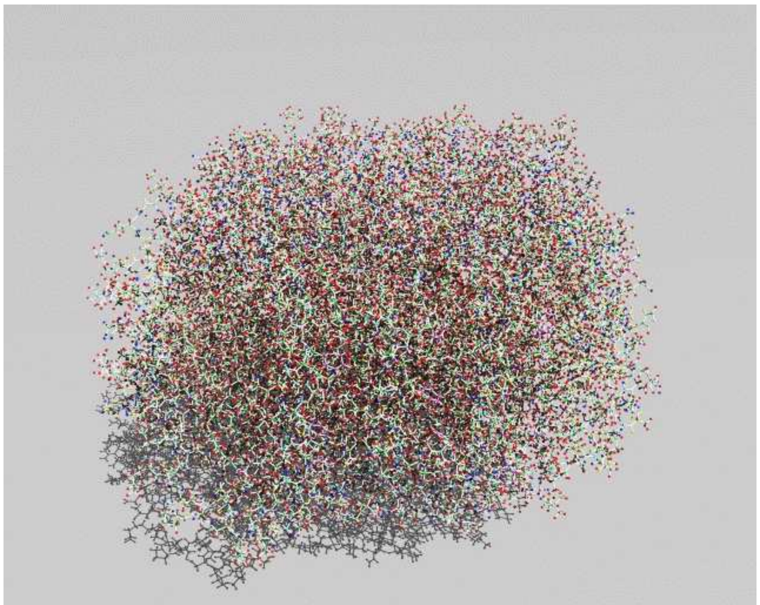


Figure 28: Apolipoprotein (ApoA1) without water, 27,850 atoms, P02647.

D.2 IM200 – Labcourse in Computational Science

During spring 2002, IM200 (a laboratory course in computational science) was given at the Department of Informatics at the University of Bergen. The main purpose of this course was to simulate and study scientific problems with the help of software packages. One of the exercises was based on the framework PROTOMOL.

The purpose of the exercise was to investigate the structure and dynamics of so-called ions trapped in an electric and magnetic field (Paul Trap). Ions in free space repel each other due to Coulomb force repulsion and expand to infinity. In a trap, an additional attractive potential keeps the system bounded. For certain parameters of the potential, the most favorable energetic configuration (the state with lowest energy) of the ions organize themselves in a linear string. Such a string is at present one of the most promising candidates for implementing a quantum processor, which have the potential of solving certain classical exponential problems in linear time.

During this exercise, the students performed several simulations with PROTOMOL and were asked to solve the following problems:

- Problem 1
  - For a two particles, calculate the equilibrium distance between the two Ca_{40}^{+} ions. $m = 40.08$, $q = 1.0$.
  - Verify your calculation by running PROTOMOL with the two-ion example. Download VMD to visualize the behavior.
  - Set up a series of simulations with increasing number of ions and run PROTOMOL until the initial string forms a bulk.
- Problem 2
  - Find the right Paul Trap parameters such that up to 41 Ca_{40}^{+} ions a convergence to a string is seen, where 42 or more ions form a zig-zag line or a bulk. Down load and use the system files (PSF, XYZ-positions and PAR) for 41 and 42 ions. For the configuration files set $\omega_z = 10^{-10}[\mathrm{fs}^{-1}]$, temperature of 10^{-6}K and use the Leapfrog integrator and a friction force.
- Problem 3

- The real scientific interest is in large numbers of particles. Run some test for different values of N and develop a model for the time complexity for the code. Assume you got the new IBM regatta system at your disposal for a week, how large system can you simulate?
- If we increase the number of ions beyond the break-over from string to bulk other forms will appear. Set the number of ions to 10000 and compute until a stable form is achieved. How does this form look like?
- Now the problem has become computationally expensive. Thus we need fast force calculation and efficient time integration. Play with different force calculation and time integrators and find one that balance efficiency with accuracy well.

A detailed description can be found at

`http://www.ii.uib.no/~matthey/im200/02/`

D.3 Magnetic holes – Ugelstad spheres

Ugelstad spheres in a magnetic fluid and an applied magnetic field create an analogy to Archimedes law. The spheres become magnetic holes and act as magnetic dipoles with a dipole moment in the opposite way of the magnetic field.

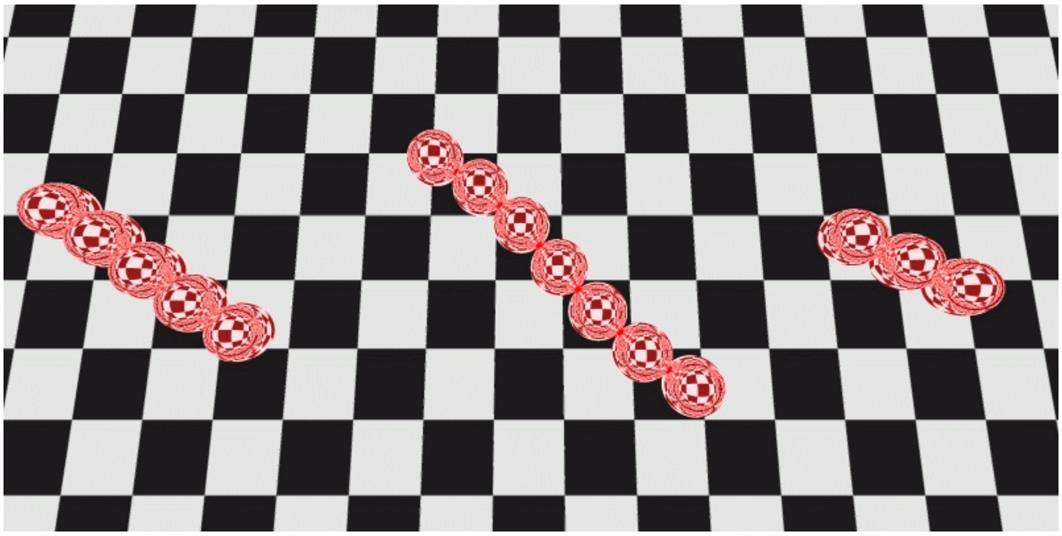


Figure 29: 15 Ugelstad spheres (by courtesy of Andreas Hellesøy).

The strength of the dipole $\vec{\sigma}$ is governed by the volume displacement of the ferrofluid, the strength of the external magnetic field and the susceptibility of the ferrofluid:

$$\vec{\sigma} = -V\chi_{eff}\vec{H}. \tag{51}$$

χ_{eff} is the effective susceptibility as a result of the geometry of the Ugelstadbead, a sphere. The force on a dipole i interacting with a dipole j of equal strength and direction is given by

$$F_i = \sum_{i>j}\left(-3\frac{\sigma^2(t)\cdot\vec{r}_{ij}}{r_{ij}^5} + 15\frac{(\vec{\sigma}(t)\cdot\vec{r}_{ij})^2\cdot\vec{r}_{ij}}{r_{ij}^7} - 6\frac{\vec{\sigma}(t)\cdot(\vec{\sigma}(t)\cdot\vec{r}_{ij})}{r_{ij}^5}\right) \tag{52}$$

In experiments comparing with the theoretical results, the spheres and the magnetic fluid are confined between two glass-plates. The boundary conditions between the ferrofluid and the glass-plates can be satisfied by introducing so-called mirror-dipoles. An infinite series of imaginary dipoles of Ugelstadspheres mirror-images would appear, if the glass-plates were mirrors satisfying the boundary conditions.

The strength of the mirror-images decreases as κ^i, where i is the image number and

$$\kappa = \frac{\chi_f}{\chi_f + 2}, \tag{53}$$

and χ_f is the susceptibility of the ferrofluid.

This work was carried out by Andreas Hellesøy[26] for his Master's degree and as part of the NOTUR Technology Transfer Project No. 5, funded in part by the Norwegian Research Council. More details can be found at

http://www.fi.uib.no/~hellesoy/TTP5.html

[26]Department of Physics, University of Bergen.

D.4 Coulomb crystals

Ion Coulomb Crystals are the solid state of plasma containing only particles of the same sign of charge. One-component plasma (OCP's) consist only of one single ion species (Figure 30). OCP's at low temperatures have been studied intensively theoretically and during the last 10 years also experimentally with laser cooled ions in traps [52, 94, 102, 116].

The potential is a sum of electrostatic repulsions and trap attractions

$$U = \sum_{i=1,j\neq i}^{N} U_{ij}^{\text{electrostatic}} + \sum_{i=1}^{N} \left(\frac{1}{2} m_i \omega_{xy}^2 (x_i^2 + y_i^2) + \frac{1}{2} m_i \omega_z^2 z_i^2 \right), \quad \vec{r}_i = (x_i, y_i, z_i)^\top . \tag{54}$$

Here, $\vec{r}_i$ is the coordinate of particle i. This ongoing project is a collaboration with Jan Petter Hansen[27] and Michael Drewsen[28] who provides experimental data. We investigate for increasing numbers of ions whether they converge to shell structure or not (Figures 31,35-38), especially for bi-crystals (Figure 32). $U^{\text{electrostatic}}$ is evaluated with the multi-grid method. For the larger simulations the multi-grid method is about 100-300 times faster with a maximum relative force error of order 10^{-3} compared to the direct method. For the convergence to shell or lattice structure this accuracy is appropriated. If needed, one may run some few final steps with a more accurate method. It is part of the NOTUR Technology Transfer Project No. 5, funded in part by the Norwegian Research Council.

[27] Department of Physics, University of Bergen.

[28] The ion Trap group, Institute of Physics and Astronomy, University of Aarhus. `http://www.ifa.au.dk/iontrapgroup/`

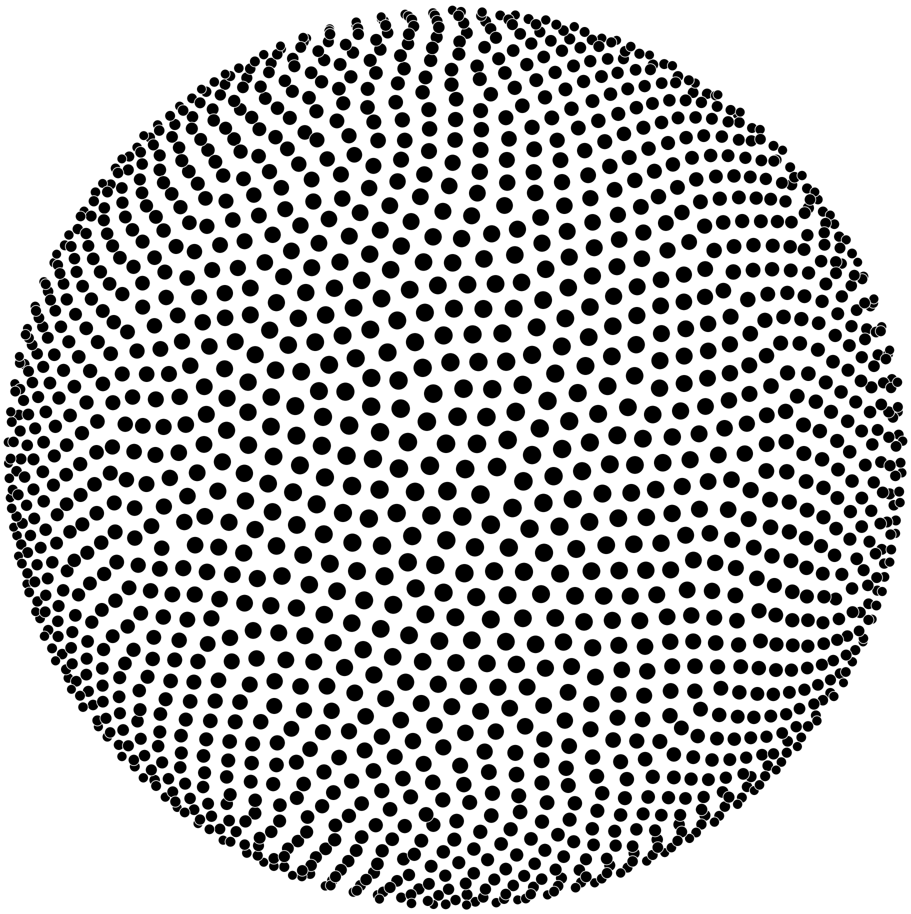


Figure 30: Front hemisphere of the outer shell of a 20,288 Ca_{40}^{+} system.

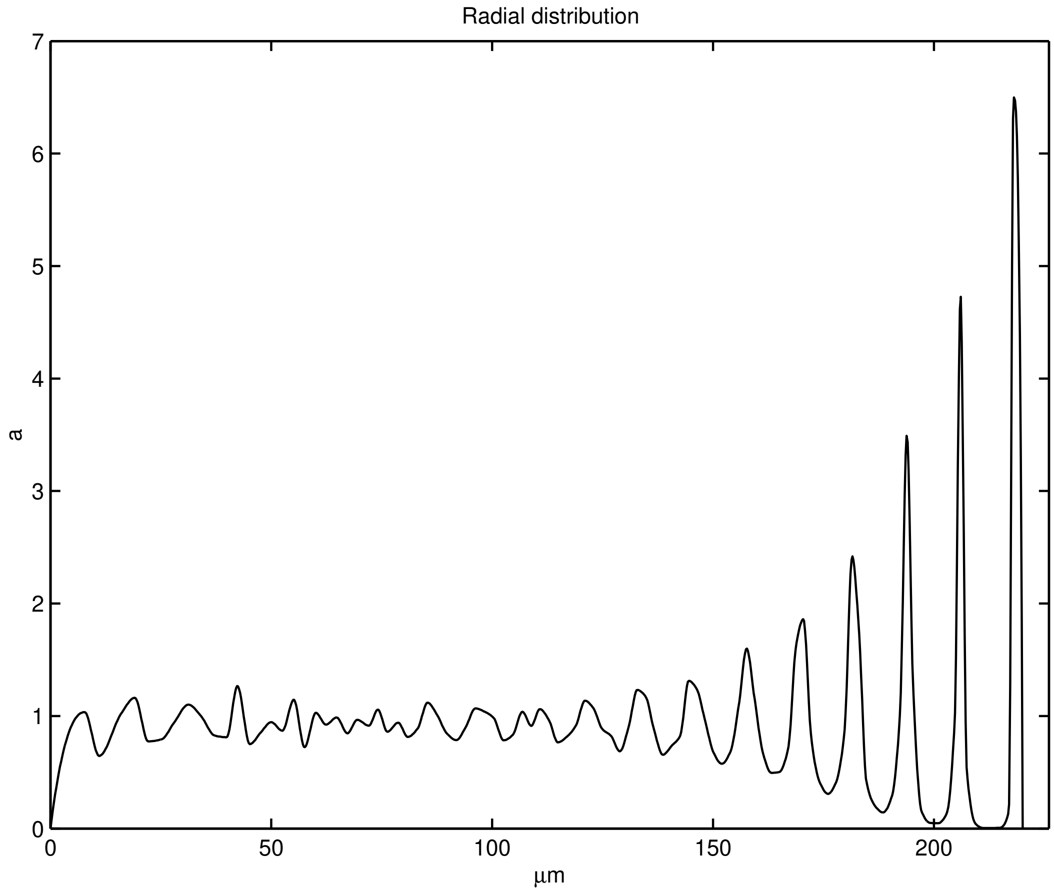


Figure 31: Radial distribution of a system with 20,288 Ca_{40}^{+}.

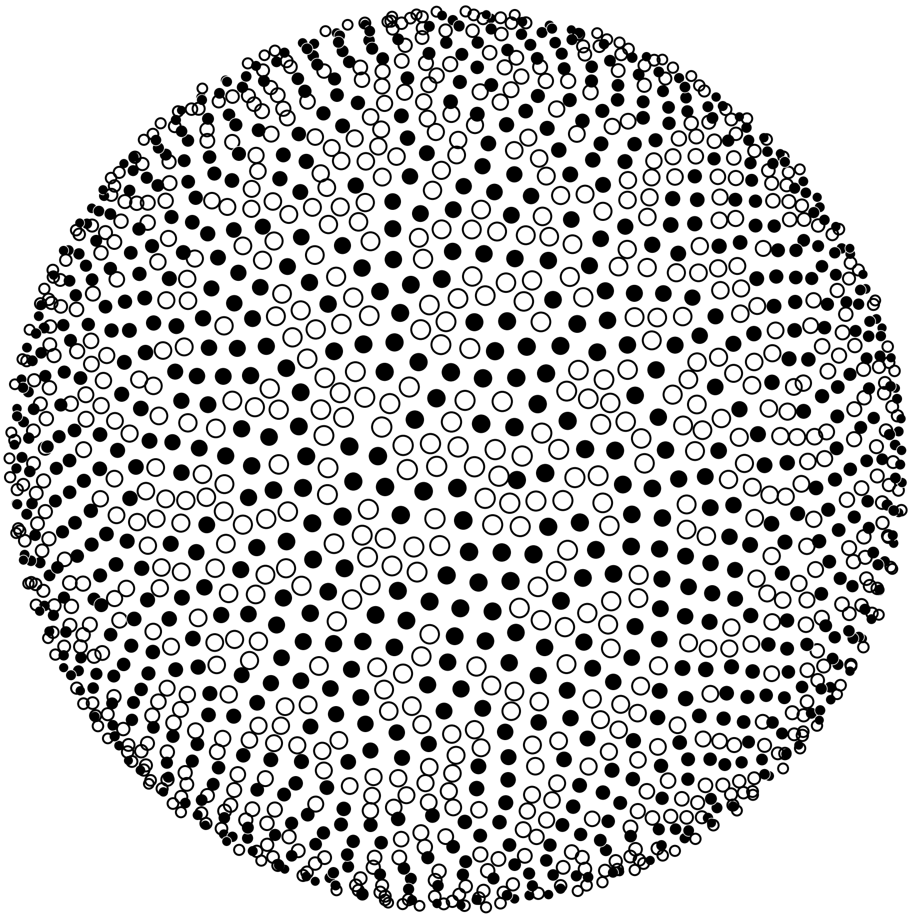


Figure 32: Front hemisphere of the outer shell of a 10,144 Ca_{40}^{+}($\circ$) and 10,144 A_{80}^{2+}($\bullet$) system.

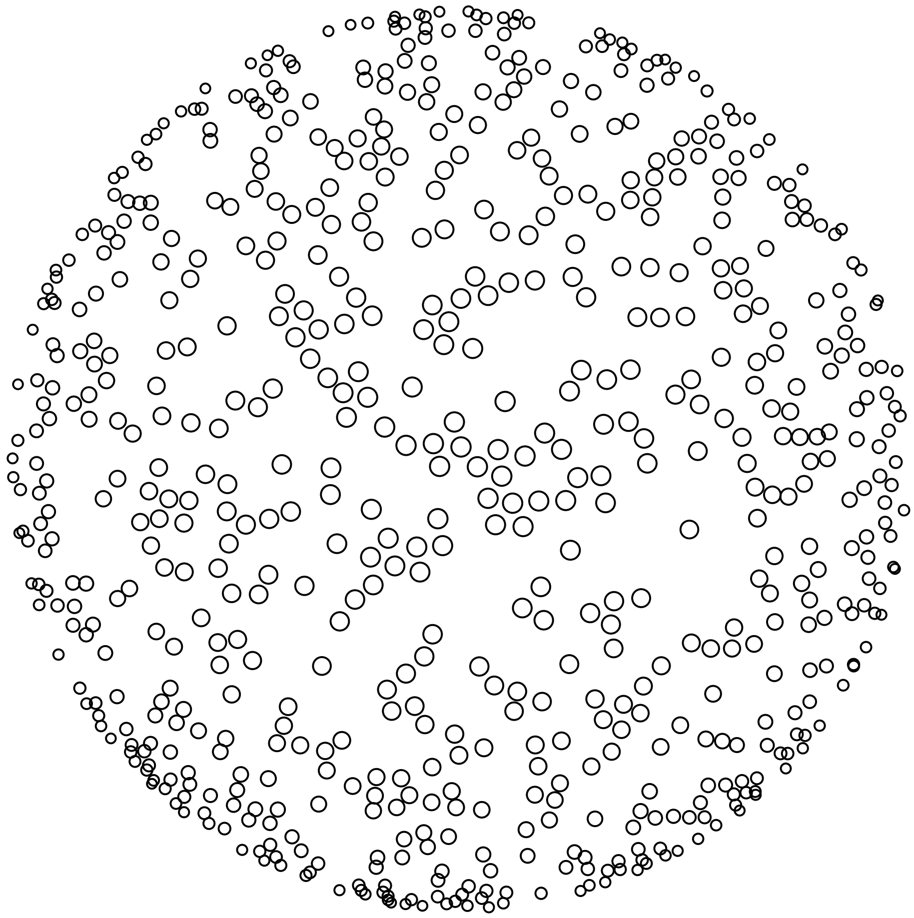


Figure 33: Front hemisphere of the outer shell (250.4-252.2μm) of a 10,144 Ca^{+}_{40}($\circ$) and 10,144 A^{2+}_{80}($\bullet$) system.

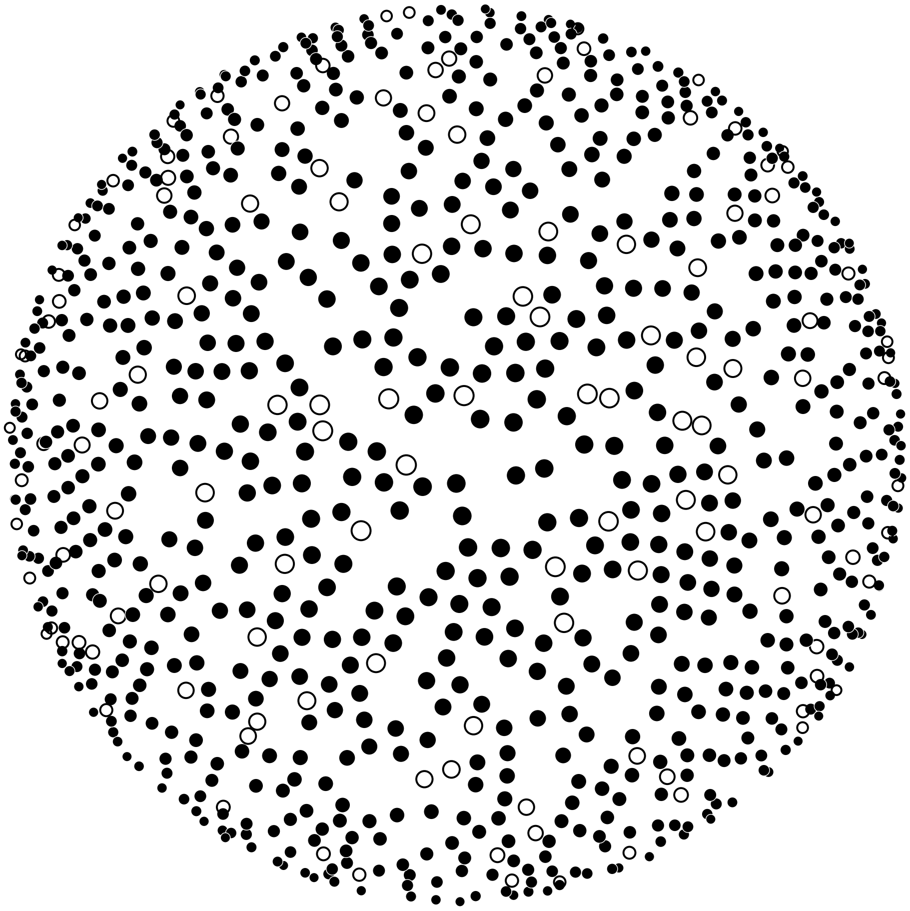


Figure 34: Front hemisphere of the outer shell (242.2-250.4μm) of a 10,144 Ca^{+}_{40}($\circ$) and 10,144 A^{2+}_{80}($\bullet$) system.

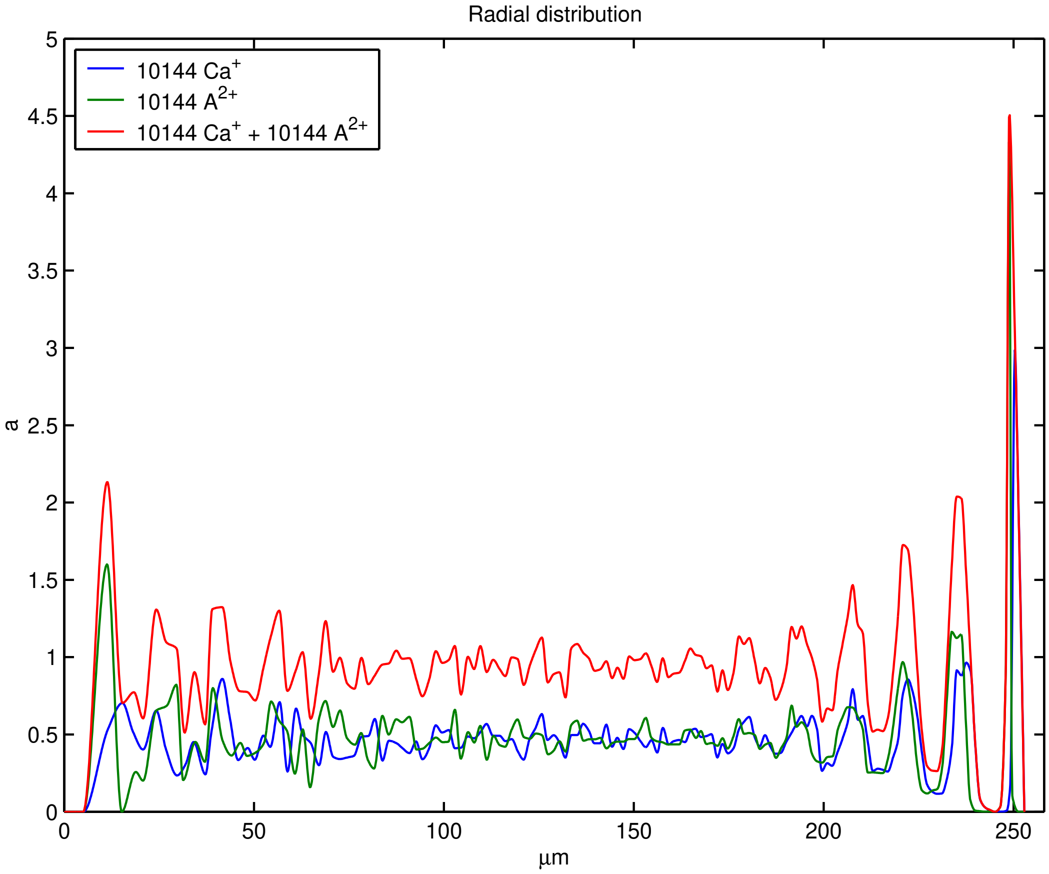


Figure 35: Radial distribution of a system with 10,144 Ca_{40}^{+} and 10,144 A_{80}^{2+}.

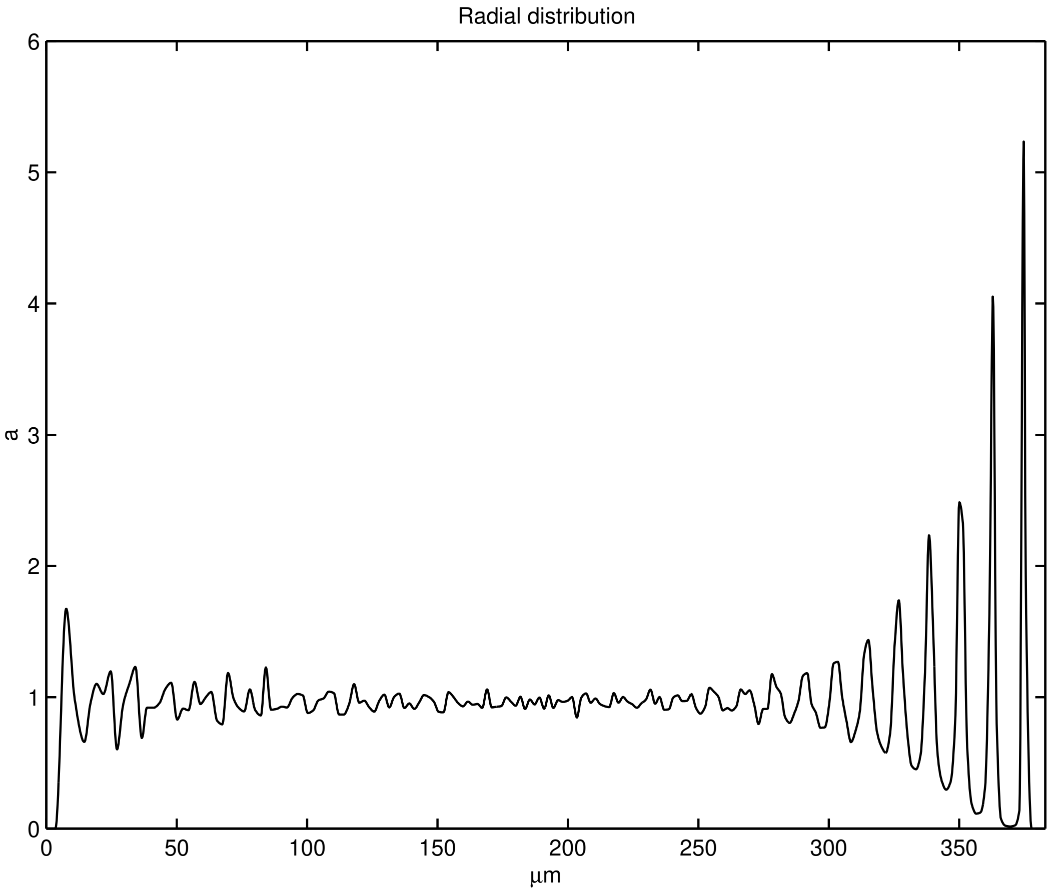


Figure 36: Radial distribution of a system with 100,000 Ca_{40}^{+}.

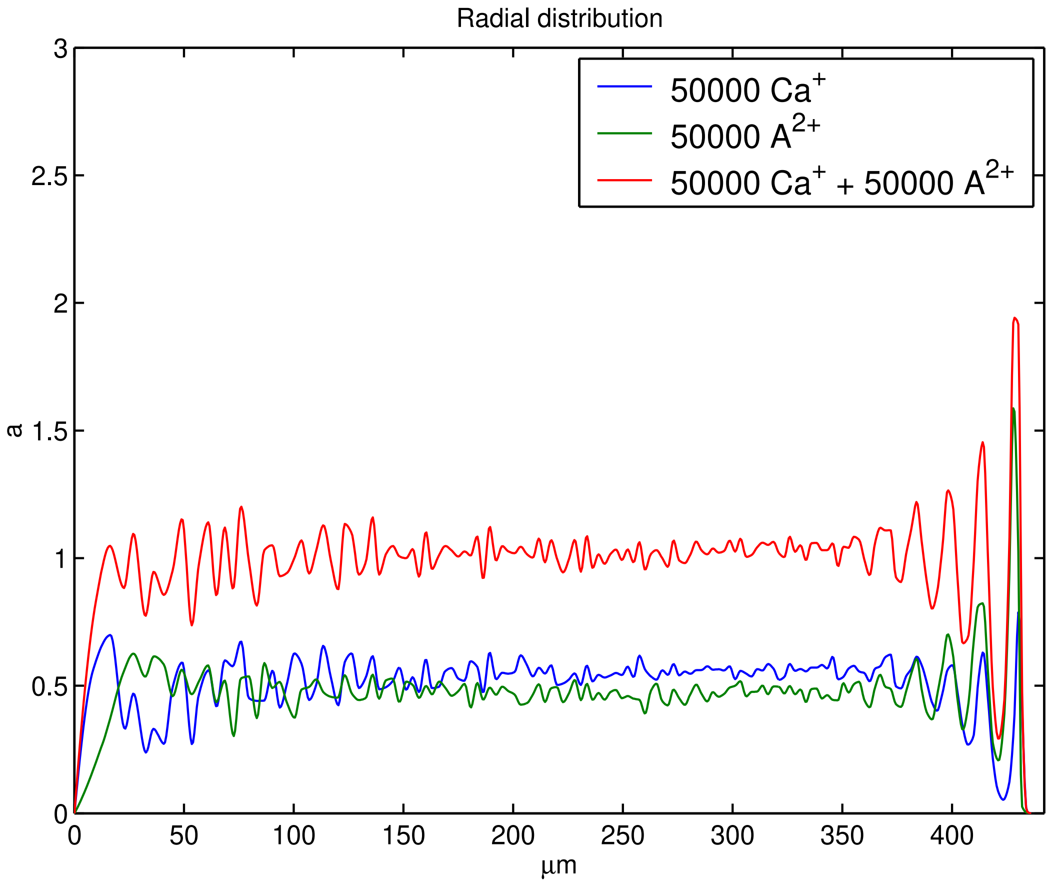


Figure 37: Radial distribution of a system with 50,000 Ca^+_{40} and 50,000 A^{2+}_{80}.

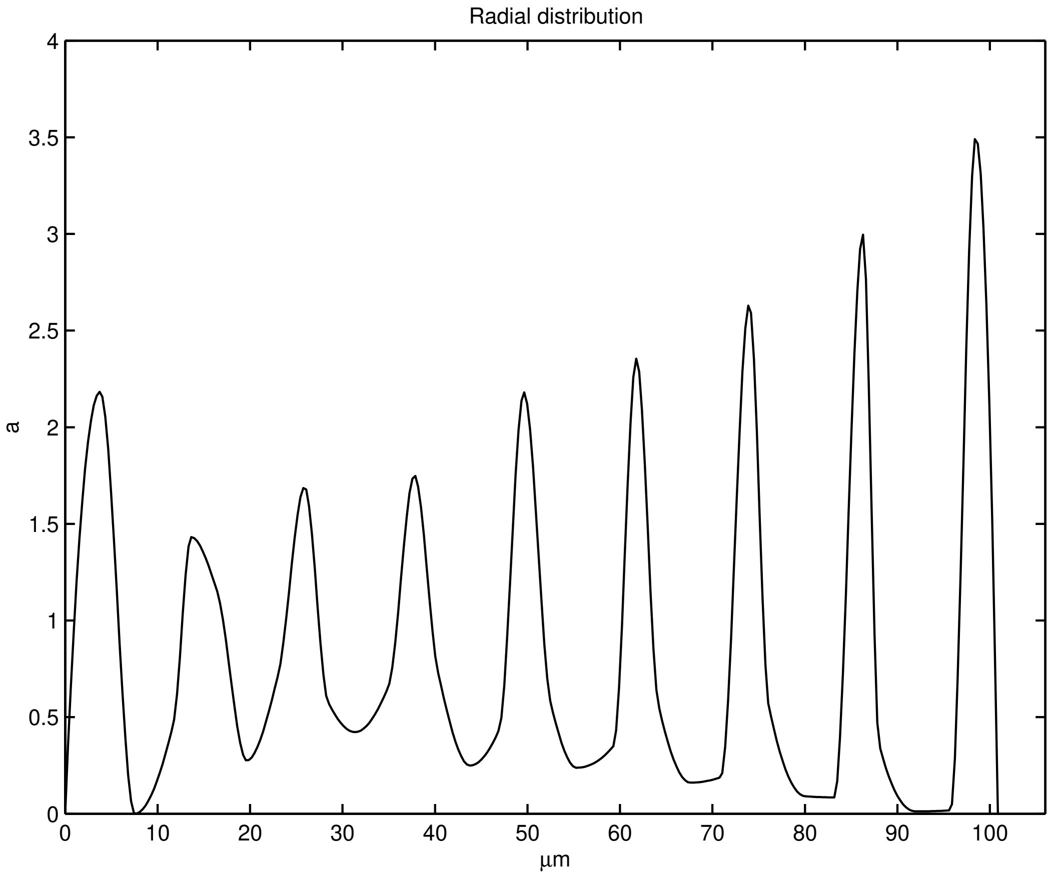


Figure 38: Radial distribution of a system with 2,057 Ca^{+}_{40}.