PROTOMOL: A Molecular Dynamics Research Framework for Algorithmic Development

T. Matthey[1], A. Ko[2], and J.A. Izaguirre[2]

[1] Department of Informatics
University of Bergen
5020 Bergen, Norway
Thierry.Matthey@ii.uib.no
[2] Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA 46556-0309
izaguirr@cse.nd.edu

Abstract. This paper describes the design and evaluation of PROTOMOL, a high performance object-oriented software framework for molecular dynamics (MD). The main objective of the framework is to provide an efficient implementation that is extensible and allows the prototyping of novel algorithms. This is achieved through a combination of generic and object-oriented programming techniques and a domain specific language. The program reuses design patterns without sacrificing performance. Parallelization using MPI is allowed in an incremental fashion. To show the flexibility of the design, several fast electrostatics (N-body) methods have been implemented and tested in PROTOMOL. In particular, we show that an $\mathcal{O}(N)$ multi-grid method for N-body problems is faster than particle-mesh Ewald (PME) for $N > 8,000$. The method works in periodic and non-periodic boundary conditions. Good parallel efficiency of the multi-grid method is demonstrated on an IBM p690 Regatta Turbo with up to 20 processors for systems with $N = 10^2, 10^4$ and 10^6. Binaries and source code are available free of charge at `http://www.nd.edu/~lcls/protomol`.

1 Introduction

Molecular dynamics (MD) is an important tool in understanding properties and function of materials at the molecular level, including biological molecules such as proteins and DNA. The challenge of MD is related to the multiple length and time scales present in systems. For example, biological molecules have thousands of atoms and time scales that span 15 orders of magnitude. The MD research community continually develops multiscale integrators, fast N-body solvers, and parallel implementations that promise to bring the study of important systems within reach of computational scientists.

Although there are many programs for MD, most are very complex and several are legacy codes. This makes it harder for algorithm developers to incorporate their algorithms and disseminate them. The complexity usually arises from parallelization and other optimizations.

PROTOMOL has been designed to facilitate the prototyping and testing of novel algorithms for MD. It provides a domain specific language that allows user prototyping

of MD simulation protocols on the fly, and it sacrifices neither performance nor parallelism. Its sequential performance is comparable to NAMD 2 [1], one of the fastest MD programs available, and it has good scalability for moderate numbers of processors.

PROTOMOL combines techniques from generic and object-oriented design in C++, but the lessons are applicable to other object oriented languages. It uses several design patterns, including some for simulations of dynamics of particles. These design pattern implementations have been sufficiently general to be reused in another framework called COMPUCELL, which models morphogenesis and other processes of developmental biology at the cellular and organism level [2].

PROTOMOL has been used in several applications, including simulation of ionic crystals, magnetic dipoles, and large biological molecules. It has also been used in courses on scientific computing simulations. As a proof of its flexibility, the design of PROTOMOL has allowed the prototyping of several new algorithms and effective implementation of sophisticated existing ones, cf. [3]. Substantial portions of this paper are in [4]. However, parallel results for multi-grid methods are reported here for the first time.

2 Physical and mathematical background

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

$$m_i \frac{d^2}{dt^2} \boldsymbol{x}_i(t) = \boldsymbol{F}_i(t), \tag{1}$$

where m_i is the mass of atom i, $\boldsymbol{x}_i(t)$ the atomic position at time t and $\boldsymbol{F}_i(t)$ the instant force on atom i. The force $\boldsymbol{F}_i$ is defined as a gradient of the potential energy

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

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

$$U = U^{\text{bonded}} + U^{\text{non-bonded}} \tag{3}$$

$$U^{\text{bonded}} = U^{\text{bond}} + U^{\text{angle}} + U^{\text{dihedral}} + U^{\text{improper}} \tag{4}$$

$$U^{\text{non-bonded}} = U^{\text{electrostatic}} + U^{\text{Lennard-Jones}}. \tag{5}$$

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^{bond}, U^{angle}, U^{dihedral} and U^{improper} define the covalent bond interactions to model flexible molecules. $U^{\text{electrostatic}}$ represents the well-known Coulomb potential and $U^{\text{Lennard-Jones}}$ models a van der Waals attraction and a hard-core repulsion.

2.1 Numerical integrators

Newton's equation of motion is a second order ordinary differential equation. Its integration is often solved by the numerical leapfrog method, which is time reversible

and symplectic. Despite its low order of accuracy, it has excellent energy conservation properties and is computationally cheap.

A complete MD simulation is described by Algorithm 1. It consists basically of the loop numerically solving the equation of motion, 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 (by a half step)
    (b) Update positions (by a full step)
    (c) **Evaluate** forces on each particle
    (d) Update velocities (by a half step)
3. Post-processing

Algorithm 1: Pseudo-code of an MD simulation.

2.2 Force evaluation

Force evaluation in MD typically consists of bonded and non-bonded interactions. Bonded interactions (U^{bond}) are short-range and comparable cheap to compute. Non-bonded interactions are of a long-range nature and determined at run-time. They are the most computationally expensive. Thus, most MD program optimizations happen here.

One of the most common optimizations for non-bonded force computations is the use of cutoffs to limit the spatial domain of pairwise interactions. Closely related to this is the use of switching functions to bring the energy and forces smoothly to zero at the cutoff point. Furthermore, cutoff computation can be accelerated through the use of cell lists or pair lists to achieve $\mathcal{O}(N)$. PROTOMOL implements generic facilities supporting all these optimizations.

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. Fast algorithms for electrostatic force evaluation implemented in PROTOMOL are described in Sections 2.2-2.2. In particular, we describe the parallel implementation of a novel multi-grid summation method (MG) for $\mathcal{O}(N)$ fast electrostatics in periodic boundary conditions or vacuum.

Ewald summation A general potential energy function U of a system of N particles with an interaction scalar function $\phi(\boldsymbol{x}_{ij} + \boldsymbol{n})$ and periodic boundary conditions can be expressed as an infinite lattice sum over all periodic images. For the Coulomb potential energy this infinite lattice sum is only conditionally convergent.

The Ewald sum method separates the electrostatic interactions into two parts: a short-range term handled in the direct sum and a long-range smooth varying term handled approximately in the reciprocal sum using Fourier transforms. This splitting

changes the potential energy from the slowly and conditionally convergent series into the sum of two rapidly converging series in direct and reciprocal space and a constant term.

This algorithm scales like $\mathcal{O}(N^2)$ unless some optimizations are performed. The splitting parameter β, which determines the relative rates of convergence between the direct and reciprocal sums, can be adjusted to reduce the computational time to $\mathcal{O}(N^{3/2})$, cf. [5].

Particle mesh Ewald Using the discrete Fast-Fourier transforms (FFT), the mesh-based Ewald methods approximate the reciprocal-space term of the standard Ewald summation by a discrete convolution on an interpolating grid. By choosing an appropriate splitting parameter β, 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. This problem has been addressed by MDSIMAID, a recommender system that proposes possible optimal choices for a given system and a required accuracy [6].

At present, there exist several implementations based on the mesh-based Ewald method, but they differ in detail. Smooth particle-mesh Ewald (SPME) [7] is implemented in PROTOMOL. The mesh-based Ewald methods are affected by errors when performing interpolation, FFT, and differentiation [5]. Accuracy increases when the interpolation order or the number of grid points increase.

Multi-grid summation Multi-grid summation (MG) has been used used to solve the N-body problems by [3, 8]. MG imposes a hierarchical separation of spatial scales and scales as $\mathcal{O}(N)$. The pair-wise interactions are *split* into a local and a smooth part. The local parts are short-range interactions, which are computed directly. The smooth part represents the slowly varying energy contributions, approximated with fewer terms – a technique known as *coarsening*. MG uses interpolation unto a grid for both the charges and the potential energies to represent its smooth – *coarse* – part. The splitting and coarsening are applied recursively and define a grid hierarchy (Figure 1). For the electrostatic energy, the *kernel* is defined by $G(r) = r^{-1}$ and $r = ||\boldsymbol{y} - \boldsymbol{x}||$. $G(r)$ 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

$$G_{\text{smooth}}^k(r) = \begin{cases} G_{s_k}(r) : r|| < s_k \\ G(r) \quad : \text{otherwise.} \end{cases} \tag{6}$$

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 $a = 2$. Corrections of the energies are required when the modified, smoothed kernel is used instead of the exact one.
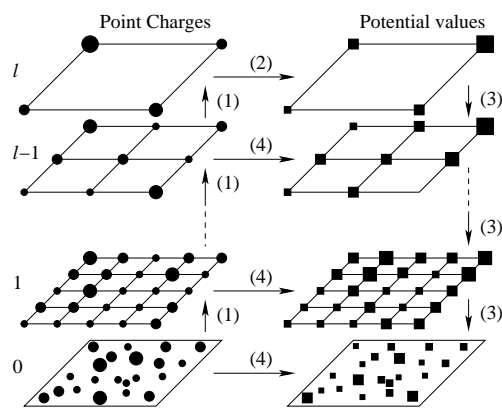
Fig. 1: The multilevel scheme of the MG algorithm. (1) Aggregate to coarser grids; (2) Compute potential energy induced by the coarsest grid; (3) Interpolate energy values from coarser grids; (4) Local corrections.
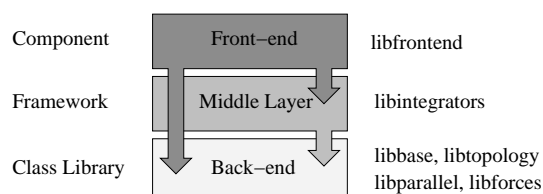
3   Framework design

MD programs may substantially differ in design, but they are essentially all based on Algorithm 1. Four 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*. MTS methods require careful fine-tuning to get the full benefit of the technique.
2. Allow developers to easily integrate and evaluate *novel force algorithms* schemes. For example, the force design allows the incorporation of sophisticated multiscale algorithms, including mesh-based methods and MG.
3. Develop an encapsulated parallelization approach, where sequential and parallel components co-exist. This way, developers are not forced to consider the distributed nature of the software. Parallelism itself is based on range computation and a hierarchical master-slave concept [9].
4. Provide facilities to compare accuracy and run-time efficiency of MD algorithms and methods.

   For the design of the component-based framework PROTOMOL, three different modules were identified. These are shown in Figure 2 and are described next. The front-end provides *components* to compose and configure MD applications. The components are responsible for composing and creating the actual MD simulation set up with its integration scheme and particle configuration. This layer is strongly decoupled from the rest to the extent that the front-end can be replaced by a scripting language.

   The middle layer is a *white-box framework* for numerical integration reflecting a general MTS design. The back-end is a *class library* carrying out the force computation and providing basic functionalities (see Section 3.1). It has a strong emphasis on run-time efficiency.

Fig. 2: The component-based framework PROTOMOL.

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, SPME, MG, etc.). The front-end is mainly the pre- and post-processing in Algorithm 1 and is not detailed in this paper, whereas the middle layer is briefly explained to give an overview of the collaboration of integrators and their associated forces. A complete domain analysis of the integrator library used in PROTOMOL is in [10], and some references to novel integrators developed and implemented using the program are [11, 12].

3.1   Force design

The forces are designed as separate components and part of the computational backend. From an MD modeling point of view and from performance considerations, five different requirements (or customizable options) are proposed. These are discussed below:

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 Cell Manager component to retrieve efficiently the spatial information of each particle. This has $\mathcal{O}(1)$ complexity.
R4 A function defining the force and energy contributions on an n-tuple.
R5 A switching function component to make the force and energy terms smoother.

Force interface In order to address these requirements, several design approaches can be chosen. To avoid an all inclusive interface with mammoth classes, we use multiple inheritance and generic programming. We combine templates and inheritance in the Policy or Strategy pattern [13, 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.

The algorithm to select the n-tuples (R1) is customized with the rest of the four requirements (R2-R5). This allows the simultaneous evaluation of different types of forces with the same algorithm. Complex force objects stem from non-bonded forces. For example, to define an electrostatic force, 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 based on a cell list algorithm (R3), defining some boundary

conditions (R2), a function defining the energy and force contributions between two arbitrary atoms. We may even modify the energy and force by specifying a switching function. The forces are designed with a common interface, a deferred feature called `evaluate(...)` that does the evaluation of the force contributions based on its parameterization and policy choices.

Force object creation Once the forces are designed and implemented, we need to create (or instantiate) the actual force objects needed by integrators. This can be solved by a sort of "just-in-time" (JIT) compiler that can transform a given input definition into a real force object.

The requirements of object creation are satisfied by the Abstract Factory [13, pp. 87-95] and the Prototype [13, pp. 117-126] patterns. The Abstract Factory pattern delegates the object creation, and the Prototype pattern allows dynamic configuration. At the end of the process, 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 [13, pp. 127-134].

3.2   Performance monitoring

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. At present, pairs of forces can be compared to determine energy and force errors of new force methods. 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. This is important to validate and verify a simulation.

For the benchmarking of forces, a timer function was implemented to measure the total and average time spent in dedicated force methods. Comparisons can be nested to evaluate accuracy and run-time performance simultaneously. The comparison of force pairs is based on the Count Proxy pattern [13, pp. 207-217] to link two forces together and calculate the actual errors.

4   Performance evaluation

We describe the performance evaluation of the fast electrostatic methods implemented in PROTOMOL and described in this paper. Figure 3 shows parallel scalability of PROTOMOL applied on Coulomb Crystal systems [14, 15], which are defined by a – computationally dominating – electrostatic part and an electric field with linear work complexity. The full electrostatics are solved by MG. The simulations were performed on an IBM p690 Regatta Turbo. Note that the sequential speedup for $N = 10^6$ is of order 10^2 or more compared to the direct method, and for lower accuracy a speedup of order 10^3 was observed.
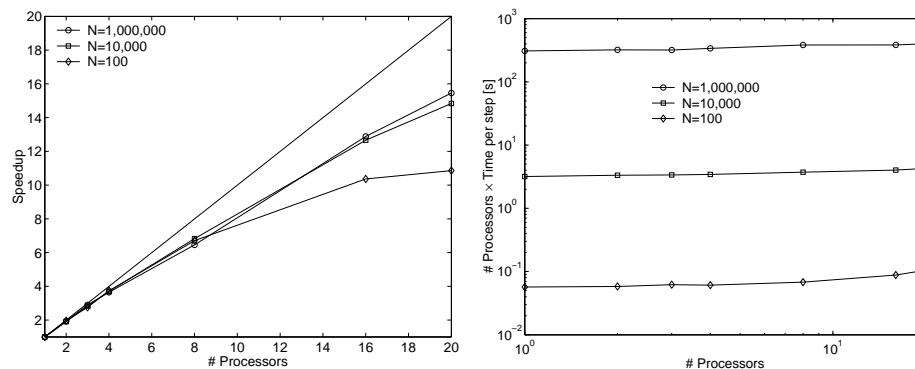
Fig. 3: Parallel scalability applied on Coulomb Crystal systems using MG electrostatic solver with relative error of order 10^{-5} or less; performed on an IBM p690 Regatta Turbo.

Our initial parallel implementation of MG is based on global communication using MPI for the smooth part defined at particle level. In Figure 1 the work of the anterpolation (1), the interpolation (3), the direct part (4) and the correction at grid level (4) can be estimated and distributed perfectly among the slaves at each level, given the grid dimensions, the softening distance and the number of particles. Accurate work estimates enables us to assign the work without any synchronization (i.e., a master node), and to reduce the idle cycles to a minimum when performing a global update of the local contributions at the grid levels. For the direct part at particle level (4) the work is distributed dynamically, since the work for a given spatial sub-domain can not be predicted statically, due to the fact that the distribution of particles is in general non-uniform. The master assigns ranges of work on demand to balance the work, which represents the force decomposition scheme with master-slave distribution. Additionally, the ranges of work are sent in pipeline to avoid slaves waiting for their next range. In general, a work range represents a collection of terms of a term of the sum U in Eq. (3). The ranges are based on the possible splittings provided by the force objects defined by the sum U. Furthermore, the work distribution can be performed using either a master-slave, as described above, or a static scheme. The static scheme distributes the work in a linear manner, which performs well for small number of nodes or uniform density systems.

Figure 4 compares the run-time of MG, which is extended to periodic boundary conditions for the first time here, and the smooth PME. The experiments are based on the TIP3 water model with atoms ranging from 1,000 to 100,000. The Ewald method is assumed to be the standard for comparison when the experiments are done in periodic boundary condition while the direct method is used for comparison when the experiments are done in vacuum. The MG method is tested both in periodic boundary conditions and in vacuum while the PME is tested in periodic boundary conditions only. The tests are performed using i686 Pentium processors running Linux. The CPU time and the relative error in evaluating the potential energy for each test are measured. The same C^2-continuous switching function and switchon of 5 Å are used for all tests.
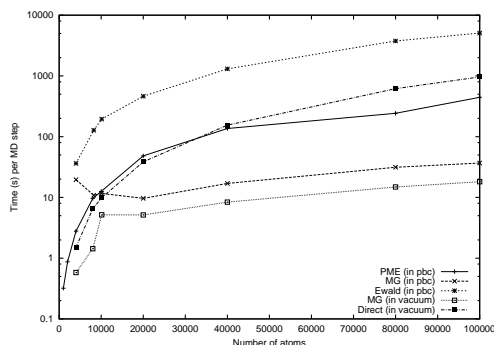
Fig. 4: Time per MD step for N-body electrostatic solvers implemented in PROTOMOL with relative error of order 10^{-4}.

The performance evaluation facilities of PROTOMOL allow us to determine the critical size and accuracy at which MG performs better than PME. MG is superior for systems of 8,000 or more particles. An optimization was performed to find the optimal parameters for each method at the lowest time and highest accuracy. Details on selection of optimal parameters for these methods are beyond the scope of this paper but can be provided on request. Some guidelines can be found in [8].

5 Discussion

The design of PROTOMOL has allowed the implementation of novel MTS integrators and fast N-body solvers. For example, the MG summation for fast electrostatic is 3-5 times faster than the particle-mesh Ewald for systems only 8,000 atoms. The parallel version of MG scales well for moderate numbers of processors: we have tested it with up to 20. Combination of these new methods have enabled simulations of million-particle systems with full electrostatics described above [15]. The facilities for performance monitoring have been very useful and general. Furthermore, the different algorithms and the comparison facilities give a unique opportunity to choose the best algorithm for a particular application and enable fair comparison of future novel algorithms.

The domain specific language makes our MD applications very flexible. By using the provide "JIT" compiler, users compose their own programs without having to touch the code. This was mainly achieved with help of the Abstract Factory pattern and the Prototype pattern, which also improves the extendibility on the developer level.

The object-oriented design of the framework along with the use of design patterns has eased the development of a fairly complex framework. By using object-oriented and generic implementation of PROTOMOL, we have achieved high performance without sacrificing extendibility. The programming language C++ has allowed us to achieve the goal of extendability, particularly we have benefited from the STL.

Acknowledgments

References

1. Kalé, L., Skeel, R., Bhandarkar, M., Brunner, R., Gursoy, A., Krawetz, N., Phillips, J., Shinozaki, A., Varadarajan, K., Schulten, K.: NAMD2: Greater scalability for parallel molecular dynamics. J. Comp. Phys. **151** (1999) 283–312
2. Chaturvedi, R., Izaguirre, J.A., Huang, C., Cickovski, T., Virtue, P., Thomas, G., Forgacs, G., Alber, M., Hentschell, G., Newman, S., Glazier, J.A.: Multi-model simulations of chicken limb morphogenesis. To appear in proceedings of the International Conference on Computational Science ICCS (2003)
3. Matthey, T.: Framework Design, Parallelization and Force Computation In Molecular Dynamics. PhD thesis, Department Of Informatics, University of Bergen (2002)
4. Matthey, T., Cickovski, T., Hampton, S., Ko, A., Ma, Q., Slabach, T., Izaguirre, J.A.: PROTOMOL: an object-oriented framework for prototyping novel algorithms for molecular dynamics. Submitted to ACM Trans. Math. Softw. (2002)
5. Petersen, H.G.: Accuracy and efficiency of the particle mesh Ewald method. J. Chem. Phys. **103** (1995)
6. Ko, A.: MDSimAid: An automatic recommender for optimization of fast electrostatic algorithms for molecular simulations. Master's thesis, University of Notre Dame, Notre Dame, IN (2002) Available from `http://www.nd.edu/~izaguirr/papers/KOthesis.pdf`.
7. Essmann, U., Perera, L., Berkowitz, M.L.: A smooth particle mesh Ewald method. J. Chem. Phys. **103** (1995) 8577–8593
8. Skeel, R.D., Tezcan, I., Hardy, D.J.: Multiple grid methods for classical molecular dynamics. J. Comp. Chem. **23** (2002) 673–684
9. Matthey, T., Izaguirre, J.A.: ProtoMol: A molecular dynamics framework with incremental parallelization. In: Proc. of the Tenth SIAM Conf. on Parallel Processing for Scientific Computing (PP01). Proceedings in Applied Mathematics, Philadelphia, Society for Industrial and Applied Mathematics (2001)
10. Izaguirre, J.A., Ma, Q., Matthey, T., Willcock, J., Slabach, T., Moore, B., Viamontes, G.: Overcoming instabilities in Verlet-I/r-RESPA with the mollified impulse method. In Schlick, T., Gan, H.H., eds.: Proceedings of 3rd International Workshop on Methods for Macromolecular Modeling. Volume 24 of Lecture Notes in Computational Science and Engineering. Springer-Verlag, Berlin, New York (2002) 146–174
11. Izaguirre, J.A., Catarello, D.P., Wozniak, J.M., Skeel, R.D.: Langevin stabilization of molecular dynamics. J. Chem. Phys. **114** (2001) 2090–2098
12. Skeel, R.D., Izaguirre, J.A.: An impulse integrator for Langevin dynamics. Mol. Phys. **100** (2002) 3885–3891
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts (1995)
14. Hasse, R.H., Avilov, V.V.: Structure and Mandelung energy of spherical Coulomb crystals. Phys. Rev. A **44** (1991) 4506–4515
15. T. Matthey, J.P.H., Drewsen, M.: Bicrystal structures in rf traps of species with identical charge-to-mass ratios. Submitted to PRL (2003)