

# A tutorial on the prototyping of multiple time stepping integrators for molecular dynamics

Jesús A. Izaguirre<sup>1</sup>, Thierry Matthey<sup>2</sup>, Jeremiah Willcock<sup>1</sup>, Qun Ma<sup>1</sup>,  
Branden Moore<sup>1</sup>, Thomas Slabach<sup>1</sup>, and George Viamontes<sup>1</sup>

<sup>1</sup> Department of Computer Science and Engineering, University of Notre Dame,  
Notre Dame, Indiana 46556-0309, USA

<sup>2</sup> Department of Informatics, University of Bergen, N-5020 Bergen, Norway

**Abstract.** This paper is intended as a practical tutorial on how to develop multiple time stepping (MTS) integrators for molecular dynamics (MD). We consider in detail how to incorporate mollified impulse methods and extrapolative MTS algorithms such as LN. It shows how object-oriented and generic programming can be used to write extendible and modular scientific software that also has high performance. This is illustrated through the design of `PROTOMOL`, an object-oriented component framework for MD. In particular, we show how `PROTOMOL` accommodates MTS integrators of arbitrary number of levels and arbitrary shape of boundary conditions, and how it encapsulates program optimizations such as parallelism. We present evidence that `PROTOMOL` achieves its design goals: (i) High performance, after evaluating `PROTOMOL` against leading MD programs, and (ii) extensibility, by demonstrating how to incorporate into `PROTOMOL` a new mollified MTS integrator that lengthens the longest time step possible for MD.

## Keywords

Molecular dynamics, multiple time stepping integrators, mollified impulse method, stability, high performance component frameworks, integrator definition language, generic programming, object composition, inheritance.

## 1 Introduction

Molecular dynamics (MD) solves Newton's equations of motion by evaluating pairwise interactions between particles (force evaluation), marching the system in time (numerical integration), and imposing boundary conditions. Different possibilities for each one of these steps present a challenge to the design of a flexible software platform to accommodate new algorithms.

This paper is intended as a practical tutorial on how to develop multiple time stepping (MTS) integrators for MD. It has a practical tone and complements other tutorials of a more mathematical nature, such as those found in [36,50,54,55,60]. We consider the requirements of MD software that incorporates state-of-the-art MTS integrators of an arbitrary number of levels. Examples of these integrators are the extrapolative method LN [4,5,52] and

the mollified impulse method, or MOLLY [25,26,37–39,61], which is in turn a more stable variant of Verlet-I/r-RESPA [27,67].

We introduce a particular algorithm-development platform for MD called PROTOMOL [40,48]. We show that it is possible to write extensible and modular software that also gives high performance. We accomplish this using object-oriented and generic programming techniques in C++, and present evidence of achieving the design goals through the following tests: (i) Extensibility, by illustrating how to incorporate MTS integrators such as Verlet-I/r-RESPA, MOLLY, and LN in a common framework, and (ii) high performance by showing performance numbers similar to that of NAMD 2.2 [41], a leading MD program with similar goals to PROTOMOL.

This tutorial should be helpful to researchers who are trying to develop new algorithms for MD. The software design ideas that are presented here should be useful to those writing their own software for MD, and may contribute towards a standardization that allows for an easier exchange of algorithms. Also, these ideas apply to languages other than C++.

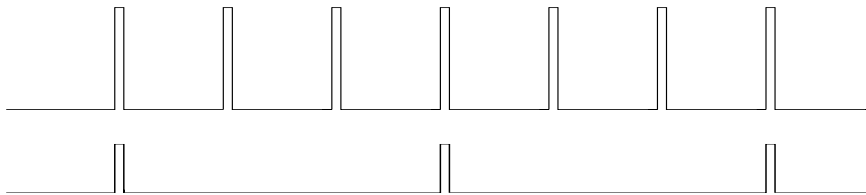
The outline of this paper is as follows: First, a review of the important components of MD is undertaken in Section 2. Then, the requirements for MD software that supports MTS integrators are identified in Section 3. Next, we introduce PROTOMOL’s design in Section 4, as a way of meeting these requirements. We present related work in Section 5. An evaluation of PROTOMOL is in Section 6. Here we present examples of implementation of several MTS integrators, along with performance evaluation. Conclusions and work in progress are in Section 7. As a service to the research community and potential users of MOLLY MTS integrators, we include an appendix showing an elegant derivation of analytical Hessians for the CHARMM force field [46]. These may prove useful in other contexts as well, such as normal mode analysis.

## 2 Molecular dynamics

Here we present a review of molecular dynamics. Besides the excellent tutorials cited in the introduction, we recommend the more extensive tutorials [9,42] and the books [23,30,45,53]. We consider MTS integrators in Section 2.1, force evaluation in Section 2.2, and boundary conditions in Section 2.3.

### 2.1 Numerical integration

The numerical integration of Newton’s equations of motion is limited by stability: the length of time steps one can take to integrate the equations of motion is fairly short relative to the total length needed for simulations — time steps are in the order of femtoseconds ( $10^{-15}$  seconds) whereas simulations of a few microseconds ( $10^{-6}$  seconds) up to one second are most desired.



**Fig. 1.** Schematic for the *Impulse* multiple time stepping method.

Multiple time stepping integrators have been used to lengthen the time step for most of the interactions in the equations of motion. These methods evaluate different parts of the force at different frequencies. Limitations on the step size in MTS integrators are still severe though, and these are due more to stability than accuracy.

A typical MTS integrator is the Verlet-I/r-RESPA multiple time stepping impulse method. In this method the force is split into different components whose dynamics correspond to different time scales, which are then represented as appropriately weighted impulses (with weights determined by consistency). The impulse method is

$$M \frac{d^2}{dt^2} X = - \sum_{n'=-\infty}^{\infty} \delta t \delta(t-n'\Delta t) \nabla U^{\text{fast}}(X) - \sum_{n=-\infty}^{\infty} \Delta t \delta(t-n\Delta t) \nabla U^{\text{slow}}(X) \quad (1)$$

where the partitioning of  $U$  into  $U^{\text{fast}}$  and  $U^{\text{slow}}$  is chosen so that an appropriate time step  $\Delta t$  for the slow part of the force is larger than a time step  $\delta t$  for the fast part. In the formula,  $\delta$  is the Dirac function. This is illustrated schematically in Fig. 1. This method permits an increase from 1 fs to 4 fs in the length of the longest time step  $\Delta t$ . MTS integrators may use more than two levels. LN is an effective method that uses three levels and stochastic damping, which allows even longer time steps than Verlet-I/r-RESPA. An elegant way to consider the generalization of MTS integrators to arbitrary numbers of levels is the use of the Trotter factorization, cf. [23,67].

When Verlet-I/r-RESPA was introduced, it was predicted that there would occur resonances that might induce instability if the frequency of the slow force impulse coincides with a normal mode frequency of the system. Resonance produces an oscillation in the positions whose amplitude increases with time. There is empirical evidence that time steps of 5 fs or greater are not possible with this method.

We have worked on the mollified impulse method (MOLLY), a family of integrators [25] that counteracts the instabilities present in the MTS Verlet-I/r-RESPA integrator. This is accomplished by perturbing the potential using time averaged positions. The time average is obtained by doing dynamics over vibrations using forces that produce those vibrations. Thus,

$$U^{\text{slow}}(X) \rightarrow U^{\text{slow}}(\mathcal{A}(X)), \quad (2)$$

with the force defined as a gradient of this averaged potential,

$$-\nabla U^{\text{slow}}(X) \rightarrow -\mathcal{A}_X(X)^T \nabla U^{\text{slow}}(\mathcal{A}(X)), \quad (3)$$

where  $\mathcal{A}_X(X)$  is a Jacobian matrix.

This perturbation compensates for finite  $\Delta t$  artifacts. Intuitively, averaged positions are better than instantaneous values for a rapidly changing trajectory  $X(t)$ . Perturbing the potential rather than the force ensures that the numerical integrator remains symplectic [53]. The force used by MOLLY is the gradient of the perturbed potential. The pre-factor  $\mathcal{A}_X(X)^T$  can be seen as a filter that eliminates components of the slow force impulse in the directions of the fast forces, and thus improves the stability of Verlet-I/r-RESPA. Different averaging functions give rise to MOLLY integrators with different stability and accuracy properties. We have used two different averagings, one based on explicit time averaging, which is reported in [61], and another based on complete elimination of linear instabilities, reported in [39]. These two methods overcome the 5 fs barrier, and the latter achieves a 50% speedup over Verlet-I/r-RESPA. A stochastic variant of MOLLY has recently been shown to allow time steps of 12 to 14 fs [38].

## 2.2 Force evaluation

Force evaluation in MD typically consists of bonded and non-bonded interactions. Bonded interactions are short-range and adequately described with simple static graph representations, and computing these forces is a simple graph traversal. Examples are bond, angle, dihedral and improper interactions. Non-bonded interactions are longer range than bonded interactions and cannot be determined statically. 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 [60, p. 149]. Furthermore, cutoff computation can be accelerated through the use of cell lists or pair lists [23, p. 368].

## 2.3 Boundary conditions

To complete the definition of the model system to be simulated one needs to specify boundary conditions which describe how the molecules interact with their surroundings. The physical characteristics of the system being simulated and the kind of questions for which answers are sought often dictate the type of boundary conditions, even though there is a certain freedom in choosing them. Two commonly used boundary conditions are:

1. Vacuum: this models isolated systems, and it is equivalent to using no boundary conditions at all. It is convenient for testing numerical algorithms and for understanding the behavior of individual macromolecules.
2. Periodic boundary conditions: this models systems with a small number of particles that experience forces as if they were in an infinitely large bulk. Periodic cells are used, and when evaluating forces multiple images of particles are considered. The shape of the original cell may be cubic or hexagonal or any other number of plane filling shapes. These forces can be summed by using the celebrated Ewald sum [16,19–21,29,43,44,49,56,63,64].

Note that any MD system should support all of the above, and several shapes in the case of periodic boundary conditions. Also, these differently shaped cells should work in conjunction with Ewald-like force evaluations and cell-list algorithms. For more details on boundary conditions the reader is referred to the book [3, pp. 24–32,156ff.].

### 3 Design challenges for MD software

The previous sections have allowed us to identify the requirements of an MD framework that is extensible and has high performance. We summarize them under two headings: component architecture and optimization encapsulation.

#### 3.1 Component architecture

It is necessary to have a modular design that allows for easy prototyping of complex methods. This may be accomplished by using a component architecture, where 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. [15]) and CORBA [2]. A component architecture for high performance software is the Common Component Architecture (CCA) [1]. The modules that we have identified, along with their functional requirements, are the following:

1. **Collaboratory front end, which performs I/O and links to visualization and steering devices.** This module needs a clean interface to the other layers. For readers interested in these considerations we recommend consulting the MD API, an MD application programmers interface that is described in [32], and the interactive and steered MD interfaces described and used in [35,65].
2. **Middle layer, which does integration of the equations of motion.**
  - (a) The main requirement here is to support an arbitrary number of levels in MTS integrators. It should be easy for the user to compose different integrators in an MTS chain.

- (b) The algorithm developer should be able to easily associate a subset of forces with each integration level.
  - (c) To accommodate MOLLY integrators, there should be a user defined pre-processing of coordinates (averaging) and a post-processing of forces (mollification).
- 3. Computational back end, which performs force evaluation and molecular state updates.**
- (a) For MTS integrators of arbitrary numbers of levels, it should allow for computation within a spatial range using different switching functions.
  - (b) For greater flexibility of the types of systems simulated, it should support arbitrarily shaped cells for periodic boundary conditions.
  - (c) To support more sophisticated integrators such as MOLLY, it should allow for the evaluation of forces at user-specified positions.

### 3.2 Encapsulation of optimizations

To satisfy the goals of extensibility and high performance, it is desirable to encapsulate optimizations such as parallelism without hurting performance. This may be achieved using generic programming mechanisms, such as *templates* in C++, cf. [57,58,66,69]. The most important optimizations implemented are described next.

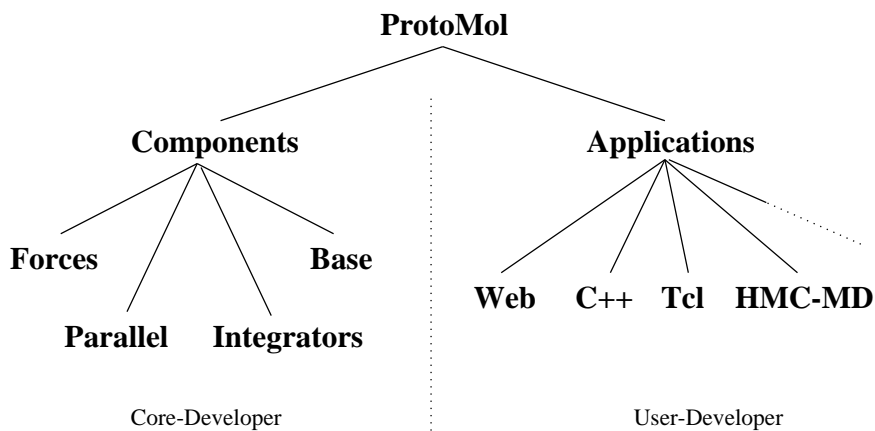
- 1. Incremental and scalable parallelization of force evaluation and integration** We hide parallelization in two objects, and thus allow for the incremental development of parallel modules. The applications developer may start with a sequential implementation of his or her algorithm, while transparently benefiting from already parallelized modules.
- 2. Cell list algorithms and boundary conditions** One would like to support cell lists for improved performance, with the following constraints:
  - (a) Arbitrarily shaped cells. This can be accomplished by making the cell lists generic (templated).
  - (b) Minimum image convention for periodic boundary conditions. This requires that the atomic pairwise distance testing be generic as well.
- 3. Fully customizable non-bonded forces using generic cells and boundary conditions**
  - (a) A visitor defining how to compute pairwise forces.
  - (b) A combined energy and force evaluation function.

## 4 ProtoMol design as a solution

PROTOMOL is an object-oriented component-based framework for MD simulations. The framework is designed for non-bonded, bonded, short-range and long-range forces for applications with tens of thousands of atoms representing biomolecules and solvents. It is designed for high flexibility, easy

extendibility and maintenance, and high performance demands, including parallelization. It is a solution to the requirements for MD software that we have identified so far.

The components (Fig. 2) implement the back end (force computation) and the middle layer (integrators), while applications (front end) use these components as basic building bricks to realize MD applications. Using the component-architecture approach makes it easy for user-developers to write their own simulations. In this section we will describe the design of the main components of PROTOMOL.



**Fig. 2.** The Framework.

#### 4.1 MTS integrators

In order to allow for user composition of MTS integrators with good performance we have implemented a twofold solution: an integrator definition language that allows the user to compose new MTS integrators at run time, and an integrator hierarchy that efficiently supports the integrator definition language. Object composition and inheritance are the main techniques for code reuse and design [24, p. 18].

**Integrator definition language.** At the highest level, our solution is to provide users with an integrator definition language, where he or she can select the following: integrator to be used at each level and forces associated with that level. An integrator can be MTS or single time stepping (STS). Different STS integrators may be used to define different equations of motion. For example, one can define constrained dynamics by using the integrator

```

Integrator {
  level N-1 integrator_name {
    cyclelength||timestep value
    force forcename forceoptions, [forcename forceoptions]
    ... }
  ...
  level 0 integrator_name {
    ... }
}

```

**Program 1:** Grammar for PROTO MOL’s integrator definition language.

called SHAKE [51], or Langevin dynamics by using the Brünger-Brooks-Karplus BBK [13] integrator. Associated with each level is a set of forces to be evaluated at that level, called a force group in our framework. This provides great flexibility to the user in partitioning the forces across the multiple levels. An abstract user definition of a new MTS integrator with N levels is given in Program 1. As examples, we present different MTS integrator definitions in Programs 2–4.

```

Integrator {
  level 1 MOLLY {
    cyclelength 6
    force Coulomb -algorithm Full -switchingFunction ComplementSWC1}
  level 0 Leapfrog {
    timestep 1 fs
    force Improper, Dihedral, Bond, Angle
    force Coulomb -algorithm Cutoff -switchingFunction SWC1
    force LennardJones -algorithm Cutoff -switchingFunction SWC1}
}

```

**Program 2:** Two level MOLLY MTS integrator.

**Integrator hierarchy and inheritance.** Now we will discuss how we support this integrator definition language in PROTO MOL. We assume throughout this presentation that one uses the velocity form of each integrator. This form can be more easily extended to multiple levels than the position form of the integrators [10,28,60,62,67]. For example, the velocity form of the Verlet or leapfrog algorithm discretizes Newton’s equations of motion as in Algorithm 1.

Verlet’s multiple time stepping extension, Verlet-I/r-RESPA can be written as Algorithm 2. Similarly, MOLLY is written as Algorithm 3. One can

```

Integrator {
  level 2 Impulse {
    cyclelength 4
    force Coulomb -algorithm Full -switchingFunction ComplementSWC1}
  level 1 Impulse {
    cyclelength 2
    force Improper, Dihedral
    force Coulomb -algorithm Cutoff -switchingFunction SWC1
    force LennardJones -algorithm Cutoff -switchingFunction SWC1}
  level 0 Leapfrog {
    timestep 1 fs
    force Bond, Angle}
}

```

**Program 3:** Three level Verlet-I/r-RESPA MTS.

```

Integrator {
  level 2 ConstantExtrapolation {
    cyclelength 50
    force Coulomb -algorithm Full -switchingFunction ComplementSWC1}
  level 1 MidpointExtrapolation {
    cyclelength 2
    force Coulomb -algorithm Cutoff -switchingFunction SWC1
    force LennardJones -algorithm Cutoff -switchingFunction SWC1}
  level 0 Leapfrog {
    timestep 0.5 fs
    force Bond, Angle, Dihedral, Improper}
}

```

**Program 4:** Three level extrapolative LN MTS integrator.

**half a kick**

$$P^{n-1+\epsilon} = P^{n-1} - \frac{\Delta t}{2} \nabla U(X^{n-1}).$$

**a drift**

$$X^n = X^{n-1} + \Delta t M^{-1} P^{n-1+\epsilon}.$$

**half a kick**

$$P^n = P^{n-1+\epsilon} - \frac{\Delta t}{2} \nabla U(X^n).$$

**Algorithm 1:** Velocity form of the **Verlet** or **leapfrog** discretization of Newton's equations of motion. The symbol  $P^{n-1+\epsilon}$  represents the momenta just after the  $(n-1)$ th kick.  $X^{n-1}$  and  $X^n$  are the positions.  $\Delta t$  is the integration time step, which is typically 1 fs in MD.

abstract the behavior of Verlet, Verlet-I/r-RESPA, and MOLLY in an algorithmic fashion as follows:

1. *halfkick()*;
2. *doDriftOrVibration()*;
3. *calculateForces()*;
4. *halfkick()*;

<p><b>half a kick</b></p> $P^{n-1+\epsilon} = P^{n-1} + \frac{\Delta t}{2} F^{\text{slow},n-1}. \quad (4)$ <p><b>a vibration</b> Propagate <math>X^{n-1}, P^{n-1+\epsilon}</math> by integrating</p> $\frac{d}{dt} X = M^{-1} P, \quad \frac{d}{dt} P = F^{\text{fast}}(X) \quad (5)$ <p>for an interval <math>\Delta t</math> to get <math>X^n, P^{n-\epsilon}</math>.</p> <p><b>half a kick</b></p> $F^{\text{slow},n} = F^{\text{slow}}(X^n), \quad (6)$ $P^n = P^{n-\epsilon} + \frac{\Delta t}{2} F^{\text{slow},n}. \quad (7)$
--

**Algorithm 2: Verlet-I/r-RESPA** method. The symbols  $P^{n-1+\epsilon}$  and  $P^{n-\epsilon}$  represent momenta just after the  $(n-1)$ th kick, and just before the  $n$ th kick, respectively.

The function *doDriftOrVibration()* is the key to the abstraction. For an MTS integrator, it executes the next level of integration, whereas for an STS integrator, it executes the drift routine. The function *calculateForces()* evaluates each force in the *force group*. MOLLY also defines a pre-processing of the positions and a post-processing of the forces. The integrator class hierarchy is designed using inheritance (Fig. 5):

1. At the base of this hierarchy there is an abstract integrator class. It provides the interface for any integrator. Every integrator defines a *run()* method that updates a set of positions and velocities by evaluating the force group. All MTS integrators have a pointer to a next integrator. The *preprocess()* and *postprocess()* methods are defined by MOLLY integrators as the averaging of positions and mollification of forces, respectively. These methods are implemented as virtual functions, where a virtual function allows for a common interface but specialized behavior. The beauty of virtual functions is that an existing code can be extended without modification.
2. An integrator that is expressed in the velocity form is what we call a standard integrator. An integrator such as LN has to be derived from a position-form integrator.

**half a mollified kick**

$$P^{n-1+\epsilon} = P^{n-1} + \frac{\Delta t}{2} F^{\text{slow}, n-1}. \quad (8)$$

**a vibration** Propagate  $X^{n-1}$ ,  $P^{n-1+\epsilon}$  by integrating

$$\frac{d}{dt} X = M^{-1} P, \quad \frac{d}{dt} P = F^{\text{fast}}(X) \quad (9)$$

(e.g., Verlet/leapfrog with time step  $\delta t$ ) for an interval  $\Delta t$  to get  $X^n$  and  $P^{n-\epsilon}$ .

**a time averaging** Calculate a temporary vector of time-averaged positions  $\bar{X}^n = \mathcal{A}(X^n)$  and a Jacobian matrix  $J^n = \mathcal{A}_x(X^n)^T$ . The time averaging function  $\mathcal{A}(x)$  uses only the fastest forces  $F^{\text{reduced}}(x)$ .

**half a mollified kick**

$$P^n = P^{n-\epsilon} + \frac{\Delta t}{2} F^{\text{slow}, n} \quad (10)$$

**Algorithm 3: Mollified impulse method.** The symbols  $P^{n-1+\epsilon}$  and  $P^{n-\epsilon}$  represent momenta just after the  $(n-1)$ th kick, and just before the  $n$ th kick, respectively. Note that  $\bar{X}^n$  is used only for the purpose of evaluating  $F^{\text{slow}}$ , it does not replace the value of  $X^n$ .

**slow force evaluation:**

$$F^{\text{slow}} := F^{\text{slow}}(X), \quad (11)$$

followed by  $k_2$  steps of {

**medium force evaluation:**

$$F^{\text{med}} := F^{\text{med}}\left(X + \frac{1}{2} \Delta t_m V\right). \quad (12)$$

followed by  $k_1$  steps of {

**half a drift:**

$$X := X + \frac{1}{2} \delta t V, \quad (13)$$

**kick:**

$$F^{\text{fast}} := F^{\text{fast}}(X), \quad F^{\text{rand}} := \sqrt{\frac{2\gamma k_B T}{\delta t}} M^{1/2} Z^n, \quad (14)$$

$$F := F^{\text{slow}} + F^{\text{med}} + F^{\text{fast}} + F^{\text{rand}}, \quad (15)$$

$$V := (1 + \gamma \delta t)^{-1} \left( V + \frac{1}{2} \delta t M^{-1} F \right). \quad (16)$$

**half a drift:**

$$X := X + \frac{1}{2} \delta t V. \quad (17)$$

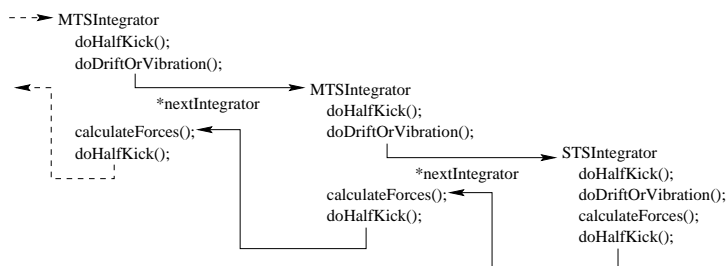
} }

**Algorithm 4:** Extrapolative three level MTS method **LN**. Assume that the medium time step  $\Delta t_m = k_1 \delta t$  and the longest time step  $\Delta t = k_2 \Delta t_m$ . Given positions  $X$  and velocities  $V$ , we show one long step of the method.

3. There are STS and MTS integrators. STS integrators do not point to another integrator, and their *doDriftOrVibration()* function performs an update of the positions. MTS integrators point to the next integrator in an integrator chain, and their *doDriftOrVibration()* function executes a number of integration steps of the next integrator in the chain.
4. The last level implements actual integrators, such as leapfrog, Verlet-I/r-RESPA, or MOLLY.

An extrapolative method such as LN (Algorithm 4) defines a new branch in the integrator hierarchy, where its base is a position-form integrator (i.e, half drift, kick, half drift). It also requires an extrapolative MTS integrator, where the forces are accumulated from one integration level to the next.

**Relationship between integrator definition language and integrator hierarchy** At run time, an integrator definition is interpreted and the correct integrator hierarchy is set up by PROTOMOL. This works because the integration methods are virtual, and are dynamically associated with the specific type of the integrator object that calls it. This does not hurt performance since integration is a relatively infrequent operation; most of the computing time is spent in force evaluation. An example of a 3-level chain of integrators set by PROTOMOL at run time is shown in Fig. 3.



**Fig. 3.** Multiple time stepping integrator.

## 4.2 Generic non-bonded forces

Our design of non-bonded forces consists of a base class defining the interface to every force evaluation, which operates upon positions, forces, and molecular structure information. The evaluation of pairwise interactions of any non-bonded force consists of:

1. An algorithm to find the pairs to be evaluated, e.g., cutoff or full range computations, etc. Closely associated with this algorithm is an algorithm to manage cell lists.
2. A switching function that smoothes the potential and force evaluation.

3. A type of boundary conditions that defines how to test distances, e.g., vacuum boundary conditions define Euclidean distance tests, whereas periodic boundary conditions define a minimum image convention.

All non-bonded forces do a distance testing, check for exclusions (if the atoms are connected through a short chain of bonds the non-bonded force between them is not computed), then call the non-bonded force function, compute the energy and its gradient, apply the switching function, and apply the chain rule to the energy and force. Thus, it is possible to reuse this mechanism for cutoff, full-range, and Ewald non-bonded force computation, and for any pairwise non-bonded force.

The great advantage of this generic design is maintenance: performance improvements to a module of the generic non-bonded force computation apply to all non-bonded forces. For example, by changing the order of cutoff and exclusion tests to perform the most frequent case first, we obtained a performance improvement between 5% and 40%. To have all non-bonded forces benefit from this optimization, it was sufficient to change the method that computes one atom-pair interaction.

Fig. 4 shows how forces and integrators communicate, where the integrators are responsible for the forces. Note that the integrator object, through its force group, executes the *evaluate()* function of bonded forces, or of the non-bonded force evaluator, for cutoff, full, or Ewald sums.

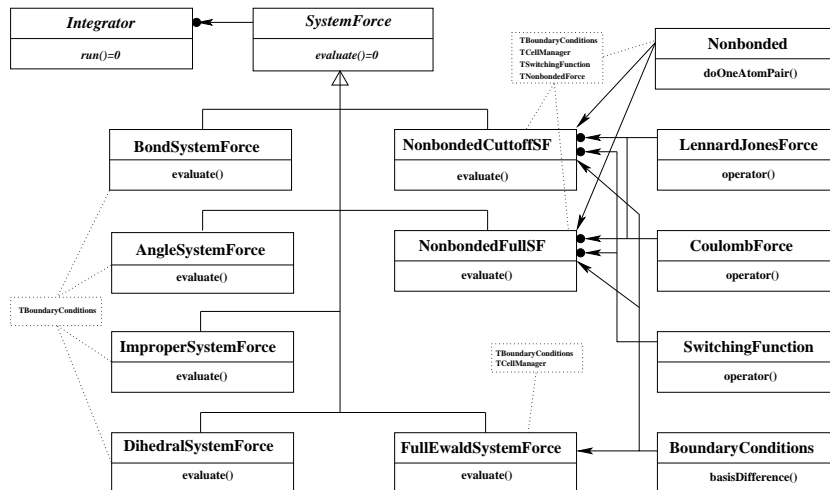
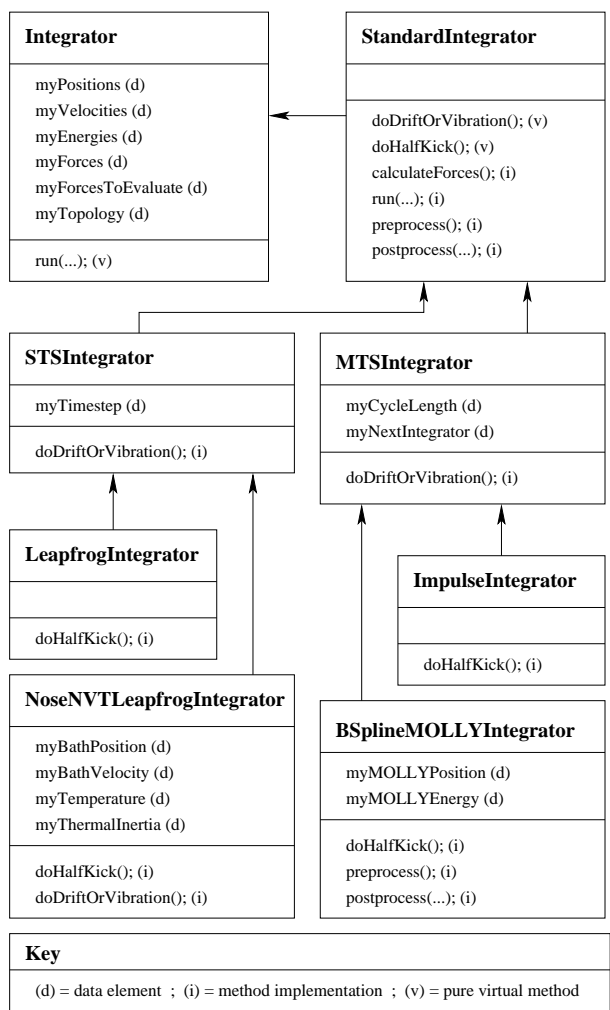


Fig. 4. Communication Between Integrators and Forces.



**Fig. 5.** Integrator hierarchy in PROTOMOL.

## 5 Related work

Besides tutorials on development of integrators for MD that have been cited above, one should cite the many excellent MD programs such as AMBER [71], CHARMM [11], NAMD [41], SPASM [8], X-PLOR [14], PINY\_MD [68], and many others [7,12,17,34,47,59]. Indeed, one has to answer the question, why another MD program? Our answer is that many design decisions of programs intended for production simulations render them inappropriate to serve as an algorithm development and academic research platform.

Of all the programs above, NAMD 2.2 addresses several of the requirements we have identified in this paper, and it has served as an inspiration in the design of PROTOMOL. NAMD has an object-oriented design, it is written in C++ and the parallel language Charm++, and has excellent performance through the use of an efficient data and work decomposition and active messages. However, its abstractions do not go far enough to allow sufficient code reuse, and many optimizations are not encapsulated enough, so that the algorithm developer has to consider them from the outset. In particular, PROTOMOL addresses the following limitations of NAMD for the development of algorithms: First, in NAMD, the integrators are hardwired into one sequence, with multiple tests to determine what kind of system one is using: for example, Langevin or constrained dynamics, full or cutoff electrostatics, and many more possibilities. Our design separates these possibilities into different modules. Also, our integrator hierarchy and the integrator definition language allow for the dynamic creation of MTS integrators, facilitating the development and testing of new algorithms. Second, in NAMD, the force computation is written using compilation macros to account for different types of non-bonded force algorithms (cutoff or full range), types of switching function, cell manager details (whether it is computing a pair within a cell or different cells), exclusions, etc. In our design we solve this problem through the use of generic force classes, as explained in the previous section. Thus, instead of compilation macros we use template parameters. Because of inlining we can still get the advantages of compiler optimizations. Third, NAMD achieves impressive scalability (being a finalist of a Gordon Bell prize in Supercomputing 2000) through the use of active messages and a combined force and spatial decomposition. These optimizations impose a great penalty on developers: one needs to deal with the distributed nature of the computations at all force levels. Also, the active message approach hides the control flow, which is now determined at run time and cannot be determined from reading the code. Our design hides these optimizations at the deepest level, and it uses common libraries such as the standard template library (STL, [57]) and MPI [22], without altering the control flow.

Finally, we acknowledge the influence of object oriented frameworks such as POOMA [31], and object oriented libraries such as OOMPAA [33], which offer examples of powerful abstractions for scientific codes. Generic libraries such as the STL [57], Blitz++ [69], and MTL [58], have taught us how to

write high performance software in C++ [66]. Excellent references to scientific computing in C++ are [6,70].

## 6 Evaluation of the framework

We show preliminary evaluation of PROTO-MOL by comparing its performance against NAMD 2.2 in Section 6.1, and by showing how to implement a MOLLY integrator in Section 6.2.

### 6.1 Performance: ProtoMol vs. NAMD 2.2

Runs were performed on a SGI Onyx2 (250MHz r10000). The times represent the wall times (in seconds) for 10 steps, sequential. For every step PROTO-MOL uses plain Ewald summation, whereas NAMD 2.2 uses Particle Mesh Ewald [18]. Table 1 shows results for a solvated BPTI system with 14281 atoms.

PROTO-MOL		NAMD2	
Periodic boundary conditions			
Plain Ewald	209.38	PME	57.10
Cutoff	33.19	Cutoff	22.53
Normal boundary conditions			
Cutoff	25.36	Cutoff	20.00
Full Coulomb	455.03	Full Coulomb	–
Full Coulomb & VdW	349.43	Full Coulomb & VdW	351.49

**Table 1.** BPTI, 14281 atoms, cutoff of 10 Å.

### 6.2 Example: B-spline mollified impulse method

Now we show how to implement a particular MOLLY integrator. We provide more detailed information in how to efficiently compute this integrator, including the analytical derivation of the angle Hessian in the appendix. This Hessian is needed in the algorithm described next. It is possible to use time averagings that consist of numerically integrating an auxiliary, reduced problem:

$$\mathcal{A}(x) = \frac{1}{\Delta t} \int_0^\infty \phi\left(\frac{t}{\Delta t}\right) \tilde{X}(t) dt \quad (18)$$

where  $\phi\left(\frac{t}{\Delta t}\right)$  is a weight function, and  $\tilde{X}(t)$  solves an *auxiliary* problem

$$M \frac{d^2}{dt^2} \tilde{X} = F^{\text{reduced}}(\tilde{X}), \quad \tilde{X}(0) = x, \quad \frac{d}{dt} \tilde{X}(0) = 0. \quad (19)$$

This approach is computationally feasible if the weight functions  $\phi$  have compact support in time. The paper [25] suggests using B-spline weight functions, which are non-zero over a short interval. The effectiveness of the averagings induced by these weight functions is directly related to the extensiveness of the time averaging. One such B-spline weight function that has been tested is called ShortAverage:

$$\phi(s) = \begin{cases} 0, & s < 0, \\ 2, & 0 \leq s < \frac{1}{2}, \\ 1, & s = \frac{1}{2}, \\ 0, & s > \frac{1}{2}. \end{cases} \quad (20)$$

The coding of  $\mathcal{A}(x)$  and  $\mathcal{A}_x(x)$  can be done by hand in a systematic manner. First the calculation of  $\mathcal{A}(x)$  is coded, and then the differentiation, applying the chain rule with respect to each of the components of  $x$  to yield code for  $\mathcal{A}_x(x)$ . As an example suppose that the leapfrog method with time step  $\delta t$  is coded for the calculation of  $\mathcal{A}(x)$ . This is then differentiated to obtain  $\mathcal{A}_x(x)$ . The result is the following code for calculating  $\mathcal{A}(x)$  and  $\mathcal{A}_x(x)$ :

Initialization is given by

$$\begin{aligned} X &:= x, X_x := I, \\ P &:= 0, P_x := 0, \\ B &:= 0, B_x := 0, \end{aligned} \quad (21)$$

and step by step integration by

$$\begin{aligned} P &:= P + \frac{1}{2}\delta t F^{\text{reduced}}(X), P_x := P_x + \frac{1}{2}\delta t F_x^{\text{reduced}}(X)X_x, \\ B &:= B + \frac{1}{2}\delta t X, B_x := B_x + \frac{1}{2}\delta t X_x, \\ X &:= X + \delta t M^{-1}P, X_x := X_x + \delta t M^{-1}P_x, \\ B &:= B + \frac{1}{2}\delta t X, B_x := B_x + \frac{1}{2}\delta t X_x, \\ P &:= P + \frac{1}{2}\delta t F^{\text{reduced}}(X), P_x := P_x + \frac{1}{2}\delta t F_x^{\text{reduced}}(X)X_x. \end{aligned} \quad (22)$$

The value  $(1/\Delta t)B$  is used for  $\mathcal{A}(x)$  and  $(1/\Delta t)B_x$  for  $\mathcal{A}_x(x)$ . We continue the above integration until we reach a value of  $t$  such that  $\phi(t/\Delta t)$  is zero at this value and remains zero for larger values of  $t$ . In practice, one would like to choose  $\delta t = \frac{\Delta t}{2}$ . If such is the case, and if ShortAverage is used, then one gets a handy equation for computing  $\mathcal{A}_x(x)$  :

$$\mathcal{A}_x(x) = I + \frac{1}{16}\Delta t^2 M^{-1}F_x(x) \quad (23)$$

where  $F_x = -U_{xx}^{\text{reduced}}(x)$ , which is the Hessian of the energies involved in the time averaging. To implement a B-spline MOLLY integrator we have to compute the Hessian matrices directly. The derivation of analytical Hessians is shown in the Appendix.

To illustrate the procedure, consider a water system. Water systems are easy to experiment with since all the water molecules are separate from each

other. Assume the O is numbered as atom 1, and the H as 2 and 3. In order to mollify the slow force impulses acting on each atom, and if only bond and angle energies are included in the reduced system, we can assemble the bond Hessians and angle Hessian to form a complete Hessian for one single molecule. Suppose the Hessian matrices of bond energy for atoms 1 and 2, and 1 and 3, *i.e.*,  $H^{\text{bd}12}$  and  $H^{\text{bd}13}$ , and the angle Hessian matrix for atoms 1 2 and 3, *i.e.*,  $H^{\text{a}123}$ .

The assembled Hessian matrix for this whole molecule is as follows:

$$H^{\text{total}} = \begin{bmatrix} H_{11}^{\text{a}123} + H_{11}^{\text{bd}12} + H_{11}^{\text{bd}13} & H_{12}^{\text{a}123} + H_{12}^{\text{bd}12} & H_{13}^{\text{a}123} + H_{12}^{\text{bd}13} \\ H_{21}^{\text{a}123} + H_{21}^{\text{bd}12} & H_{22}^{\text{a}123} + H_{22}^{\text{bd}12} & H_{23}^{\text{a}123} \\ H_{31}^{\text{a}123} + H_{21}^{\text{bd}13} & H_{32}^{\text{a}123} & H_{33}^{\text{a}123} + H_{22}^{\text{bd}13} \end{bmatrix}. \quad (24)$$

Substituting  $F_x(x)$  in Equation (23) with  $-H^{\text{total}}$ , one gets the the filter  $\mathcal{A}_x(x)$ .

## 7 Conclusions and work in progress

This paper has identified important requirements for the development of better multiple time stepping integrators, which may accommodate an arbitrary number of levels and run time flexibility in the assignment of forces to different integration levels. In particular, we have identified the requirements to accommodate Verlet-I/r-RESPA, MOLLY, and LN methods in a common framework.

We have shown the usefulness of software engineering for scientific computing, in particular the ideas of object-oriented design, particularly object composition and inheritance, which allow for a separation among modules, and the specialization of integrators according to their taxonomy; generic programming, which served to abstract common operations in the non-bonded force, allowing for parameterized routines on the non-bonded algorithm used, switching function, cell list algorithm, and boundary conditions; and object oriented design, using `.` High performance is obtained by using inlining and generic programming, since the code can be efficiently optimized by modern compilers.

We have kept the code relatively straightforward for algorithm development by encapsulating all optimizations: for example, parallelism is hidden inside a `Parallel` object, cell management for arbitrarily shaped cells is also encapsulated, as well as boundary conditions. These optimizations are usually the culprits for lack of extensibility in MD codes.

We have evaluated our framework `PROTOMOL` by illustrating how to implement complicated MTS algorithms such as a B-spline MOLLY method, and evaluated its performance by comparing to a leading MD program, `NAMD 2.2`. The paper provides enough detail to enable the reader to implement different MTS integrators, including MOLLY methods, either using

our design ideas and interfaces, or by directly using our framework. Regarding performance, although we are somewhat slower than NAMD, there is great room for improvement in the optimizations, and the flexibility of our platform is much greater. As far as parallelism, in another paper we report a parallel efficiency of 75% in 32 nodes where only non-bonded forces were parallelized [48]. It does not compare with NAMD, which to the best of our knowledge is one of the most scalable MD programs, but we provide an attractive algorithm-development and production platform.

We are making the design even more flexible by abstracting the force computation and molecular structure even more, so that a certain graph-like data structure may be used as the basis for traversals with generic visitors that actually compute the forces. We are also extending the framework to include Monte Carlo and Hybrid Monte Carlo applications.

## Acknowledgments

We acknowledge the inspiration provided by Dr. Robert Skeel’s work on MTS integrators in many aspects of this work. We are grateful to Dr. Atul Bahel for his implementation of Nosé-Hoover Verlet into PROTOMOL. The following students have collaborated in the implementation of PROTOMOL: Trevor Cickovski, Thomas Steinbach, Scott Stender, Jeffrey Stine, and Joseph Taylor. This research was partially supported under NSF Biocomplexity grant 0083653. Also, an author was supported by a fellowship from the Center for Applied Mathematics at the University of Notre Dame. Performance evaluation of PROTOMOL and NAMD was performed at the Norwegian super-computing facilities in Bergen through a Norges Forskningsråd grant.

## Appendix Derivation of the Hessian of the Angle Energy

We derive the Hessian of the angle energy used in the CHARMM force field [46]. This is the most complicated, and the same approach can be used for other bonded forces. Angle interactions describe angular bonds between three atoms. These bonds are modeled as harmonic angular springs. The energy of such a bond between atoms  $i$ ,  $j$ , and  $k$  is given by:

$$E_{angle} = E_{\theta} + E_{ub}, \quad (25)$$

$$E_{\theta} = k_{\theta} (\theta - \theta_0)^2, \quad (26)$$

$$E_{ub} = k_{ub} (|\mathbf{r}_{ik}| - r_{ub})^2, \quad (27)$$

where  $k_{\theta}$  is the force constant,  $\theta = \cos^{-1} \left( \frac{\mathbf{r}_{ij} \cdot \mathbf{r}_{kj}}{|\mathbf{r}_{ij}| |\mathbf{r}_{kj}|} \right)$ ,  $\theta_0$  is the rest angle of this bond,  $k_{ub}$  is Urey-Bradley constant,  $\mathbf{r}_{ik} = \mathbf{r}_k - \mathbf{r}_i$ ,  $|\mathbf{r}_{ik}|$  is the calculated

distance between atoms  $i$  and  $k$  and  $r_{\text{ub}}$  is rest distance for the Urey-Bradley term.

By definition, the Hessian is the second derivative of the energy function. The Hessian of the Urey-Bradley energy as shown in Equation (27) can be obtained easily.

$$E_{rr}^{\text{ub}} = 2k_{\text{ub}} \begin{bmatrix} I & 0 & -I \\ 0 & 0 & 0 \\ -I & 0 & I \end{bmatrix} + \frac{2k_{\text{ub}}r_{\text{ub}}}{|\mathbf{r}_{ik}|} \begin{bmatrix} v_{ik}v_{ik}^T - I & 0 & -r_{ik}r_{ik}^T + I \\ 0 & 0 & 0 \\ -v_{ik}v_{ik}^T + I & 0 & r_{ik}r_{ik}^T - I \end{bmatrix}. \quad (28)$$

Let  $C(\alpha, \beta, \gamma) = \frac{\alpha+\beta-\gamma}{2\sqrt{\alpha}\sqrt{\beta}}$  where  $\alpha$ ,  $\beta$  and  $\gamma$  are scalars, and  $\alpha = |\mathbf{r}_{ij}|^2$ ,  $\beta = |\mathbf{r}_{kj}|^2$ , and  $\gamma = |\mathbf{r}_{ki}|^2$ .

Now the angle energy can be expressed as

$$E_{\theta}(C(\alpha, \beta, \gamma)) = k_{\theta}(\cos^{-1}(C(\alpha, \beta, \gamma)) - \theta_0)^2. \quad (29)$$

The second derivative of the  $E_{\theta}$  part is then expressed as follows:

$$E_{rr}^{\theta} = \overbrace{E_C C_{rr}}^{E_{rr}^a} + \underbrace{\left( \frac{2k_{\theta}[\sin\theta - (\theta - \theta_0)\cos\theta]}{\sin^3\theta} \right)}_{E_{rr}^b} C_r C_r^T \quad (30)$$

where  $E_C = -\frac{2k_{\theta}(\theta - \theta_0)}{\sin\theta}$ ,  $C_r = f\alpha_r + g\beta_r + h\gamma_r$  in which  $f = \frac{\alpha - \beta + \gamma}{4\alpha^{3/2}\sqrt{\beta}}$ ,  $g = \frac{-\alpha + \beta + \gamma}{4\sqrt{\alpha}\beta^{3/2}}$ ,  $h = -\frac{1}{2\sqrt{\alpha}\sqrt{\beta}}$ ,  $\alpha_r = (-2, 2, 0)^T \sqrt{\alpha} r_{ij}$ ,  $\beta_r = (0, 2, -2)^T \sqrt{\beta} r_{kj}$ , and  $\gamma_r = (2, 0, -2)^T \sqrt{\gamma} r_{ki}$ . The  $C_{rr}$  is computed as follows:

$$C_{rr} = \overbrace{(f\alpha_{rr} + g\beta_{rr} + h\gamma_{rr})}^{C_{rr}^a} + \underbrace{(\alpha_r f_r^T + \beta_r g_r^T + \gamma_r h_r^T)}_{C_{rr}^b}, \quad (31)$$

where

$$\alpha_{rr} = \begin{bmatrix} 2I & -2I & 0 \\ -2I & 2I & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \beta_{rr} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2I & -2I \\ 0 & -2I & 2I \end{bmatrix}, \quad \gamma_{rr} = \begin{bmatrix} 2I & 0 & -2I \\ 0 & 0 & 0 \\ -2I & 0 & 2I \end{bmatrix} \quad (32)$$

and

$$\begin{aligned} f_r &= f_{\alpha} \alpha_r + f_{\beta} \beta_r + f_{\gamma} \gamma_r, & g_r &= g_{\alpha} \alpha_r + g_{\beta} \beta_r + g_{\gamma} \gamma_r, \\ h_r &= h_{\alpha} \alpha_r + h_{\beta} \beta_r + h_{\gamma} \gamma_r \end{aligned} \quad (33)$$

where

$$f_{\alpha} = \frac{-\alpha + 3\beta - 3\gamma}{8\alpha^{5/2}\sqrt{\beta}}, \quad f_{\beta} = \frac{-\alpha - \beta - \gamma}{8\alpha^{3/2}\beta^{3/2}}, \quad f_{\gamma} = \frac{1}{4\alpha^{3/2}\sqrt{\beta}} \quad (34)$$

$$g_{\alpha} = f_{\beta}, \quad g_{\beta} = \frac{3\alpha - \beta - 3\gamma}{8\sqrt{\alpha}\beta^{5/2}}, \quad g_{\gamma} = \frac{1}{4\sqrt{\alpha}\beta^{3/2}}, \quad (35)$$

$$h_{\alpha} = f_{\gamma}, \quad h_{\beta} = g_{\gamma}, \quad h_{\gamma} = 0. \quad (36)$$

In Equation (31), the first part,  $C_{rr}^a$ , becomes

$$C_{rr}^a = \begin{bmatrix} 2(f+h)I & -2fI & -2hI \\ -2fI & 2(f+g)I & -2gI \\ -2hI & -2gI & 2(g+h)I \end{bmatrix}, \quad (37)$$

whereas the second part,  $C_{rr}^b$ , becomes

$$C_{rr}^b = f_\alpha \alpha_r \alpha_r^T + f_\beta \alpha_r \beta_r^T + f_\gamma \alpha_r \gamma_r^T + g_\alpha \beta_r \alpha_r^T + g_\beta \beta_r \beta_r^T + g_\gamma \beta_r \gamma_r^T + h_\alpha \gamma_r \alpha_r^T + h_\beta \gamma_r \beta_r^T + h_\gamma \gamma_r \gamma_r^T \quad (38)$$

where

$$\alpha_r \alpha_r^T = \begin{bmatrix} 4r_{ij}\hat{r}_{ij}^T & -4r_{ij}\hat{r}_{ij}^T & 0 \\ -4r_{ij}\hat{r}_{ij}^T & 4r_{ij}\hat{r}_{ij}^T & 0 \\ 0 & 0 & 0 \end{bmatrix} \alpha \quad (39)$$

$$\beta_r \beta_r^T = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 4r_{jk}\hat{r}_{jk}^T & -4r_{jk}\hat{r}_{jk}^T \\ 0 & -4r_{jk}\hat{r}_{jk}^T & 4r_{jk}\hat{r}_{jk}^T \end{bmatrix} \beta \quad (40)$$

$$\gamma_r \gamma_r^T = \begin{bmatrix} 4r_{ik}\hat{r}_{ik}^T & 0 & -4r_{ik}\hat{r}_{ik}^T \\ 0 & 0 & 0 \\ -4r_{ik}\hat{r}_{ik}^T & 0 & 4r_{ik}\hat{r}_{ik}^T \end{bmatrix} \gamma \quad (41)$$

$$\beta_r \alpha_r^T = \begin{bmatrix} 0 & 0 & 0 \\ -4r_{kj}\hat{r}_{ij}^T & 4r_{kj}\hat{r}_{ij}^T & 0 \\ 4r_{kj}\hat{r}_{ij}^T & -4r_{kj}\hat{r}_{ij}^T & 0 \end{bmatrix} \sqrt{\alpha} \sqrt{\beta} \quad (42)$$

$$\gamma_r \alpha_r^T = \begin{bmatrix} -4r_{ki}\hat{r}_{ij}^T & 4r_{ki}\hat{r}_{ij}^T & 0 \\ 0 & 0 & 0 \\ 4r_{ki}\hat{r}_{ij}^T & -4r_{ki}\hat{r}_{ij}^T & 0 \end{bmatrix} \sqrt{\alpha} \sqrt{\gamma} \quad (43)$$

$$\gamma_r \beta_r^T = \begin{bmatrix} 0 & 4r_{ki}\hat{r}_{kj}^T & -4r_{ki}\hat{r}_{kj}^T \\ 0 & 0 & 0 \\ 0 & -4r_{ki}\hat{r}_{kj}^T & 4r_{ki}\hat{r}_{kj}^T \end{bmatrix} \sqrt{\beta} \sqrt{\gamma} \quad (44)$$

$$\alpha_r \beta_r^T = (\beta_r \alpha_r^T)^T, \quad \alpha_r \gamma_r^T = (\gamma_r \alpha_r^T)^T, \quad \beta_r \gamma_r^T = (\gamma_r \beta_r^T)^T. \quad (45)$$

The second part of Equation (30) becomes

$$E_{rr}^b = \left( \frac{2k_\theta [\sin \theta - (\theta - \theta_0) \cos \theta]}{\sin^3 \theta} \right) (f^2 \alpha_r \alpha_r^T + g f \beta_r \alpha_r^T + h f \gamma_r \alpha_r^T + g f \alpha_r \beta_r^T + g^2 \beta_r \beta_r^T + h g \gamma_r \beta_r^T + h f \alpha_r \gamma_r^T + g h \beta_r \gamma_r^T + h^2 \gamma_r \gamma_r^T). \quad (46)$$

## References

1. *Common Component Architecture Forum*. See <http://www.acl.lanl.gov/cca-forum>.
2. *The Common Object Request Broker: Architecture and Specification (Draft)*, December 1991. Revision 1.1.
3. M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, Oxford, New York, 1987. Reprinted in paperback in 1989 with corrections.
4. E. Barth and T. Schlick. Extrapolation versus impulse in multiple-timestepping schemes. II. Linear analysis and applications to Newtonian and Langevin dynamics. *J. Chem. Phys.*, 109(5):1633–1642, Aug 1998.
5. E. Barth and T. Schlick. Overcoming stability limitations in biomolecular dynamics. I. Combining force splitting via extrapolation with Langevin dynamics in LN. *J. Chem. Phys.*, 109(5):1617–1632, August 1998.
6. J. J. Barton and L. R. Nackman. *Scientific and Engineering C++: an introduction with advanced techniques and examples*. Addison-Wesley, Reading, Massachusetts, 1994.
7. D. M. Beazley and P. S. Lomdahl. Message-passing multi-cell molecular dynamics on the connection machine 5. *Parallel Computing*, 20:173–195, 1994.
8. D. M. Beazley and P. S. Lomdahl. Lightweight computational steering of very large scale molecular dynamics simulations. In *Proceedings of Supercomputing '96*, 1996.
9. H. J. C. Berendsen. Molecular dynamics simulations: The limits and beyond. In P. Deuffhard, 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 3–36. Springer-Verlag, Nov. 1998.
10. J. J. Biesiadecki and R. D. Skeel. Dangers of multiple-time-step methods. *J. Comput. Phys.*, 109(2):318–328, Dec. 1993.
11. B. R. Brooks and M. Hodošček. Parallelization of CHARMM for MIMD machines. *CDA*, 7:16–22, Dec. 1992.
12. 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.
13. A. Brünger, C. B. Brooks, and M. Karplus. Stochastic boundary conditions for molecular dynamics simulations of ST2 water. *Chem. Phys. Lett.*, 105:495–500, 1982.
14. A. T. Brünger. *X-PLOR, Version 3.1: A System for X-ray Crystallography and NMR*. Yale University Press, New Haven and London, 1992.
15. D. Chappell. *Understanding ActiveX and OLE*. Microsoft Press, 1997.
16. 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.
17. T. W. Clark, R. v. 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, Calif., 1994. IEEE Computer Society Press.
18. T. Darden, D. York, and L. Pedersen. Particle mesh Ewald. An  $N \log(N)$  method for Ewald sums in large systems. *J. Chem. Phys.*, 98:10089–10092, 1993.

19. H. Dufner, S. M. Kast, J. Brickmann, and M. Schlenkrich. Ewald summation versus direct summation of shifted-force potentials for the calculation of electrostatic interactions in solids: A quantitative study. *J. Comp. Chem.*, 18:660–676, 1997.
20. P. Ewald. Die berechnung optischer und elektrostatischer gitterpotentiale. *Ann. Phys.*, 64:253–287, 1921.
21. T. Forester and W. Smith. On multiple time-step algorithms and the Ewald sum. *Mol. Sim.*, 13(3):195–204, 1994.
22. M. Forum. MPI-2: Extensions to the Message-Passing Interface. <http://www.mpi-forum.org/docs/docs.html>.
23. D. Frenkel and B. Smit. *Understanding Molecular Simulation: From Algorithms to Applications*. Academic Press, 1996.
24. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.
25. 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, Oct. 20, 1998.
26. 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.
27. H. Grubmüller. Dynamiksimulation sehr großer Makromoleküle auf einem Parallelrechner. Master's thesis, Physik-Dept. der Tech. Univ. München, Munich, 1989.
28. 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.
29. A. Grzybowski, E. Gwozdz, and A. Brodka. Ewald summation of electrostatic interactions in molecular dynamics of a three-dimensional system with periodicity in two directions. *Phys. Rev.*, 61:6706–6712, 2000.
30. J. M. Haile. *Molecular Dynamics Simulation*. John Wiley and Sons, 1992.
31. S. Haney and J. Crotinger. How templates enable high-performance scientific computing in C++. *Computing In Science & Engineering*, 1(4):66–72, Jul-Aug 1999. POOMA reference.
32. 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/doc.ps.gz>.
33. G. A. Huber and J. A. McCammon. OOMPAA—Object-oriented model for probing assemblages of atoms. *J. Comput. Phys*, 151(1):264–282, May 1, 1999.
34. 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.
35. B. Isralewitz, M. Gao, and K. Schulten. Steered molecular dynamics and mechanical functions of proteins. *Curr. Opinion Struct. Biol.*, 2001. Invited Article.
36. 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. Available online via <http://www.cs.uiuc.edu/research/tech-reports.html>.

37. 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*. IOS Press, Amsterdam, Netherlands, Jan 2001.
38. 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, Feb. 1, 2001.
39. 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.
40. J. A. Izaguirre, J. Willcock, T. Matthey, T. B. Slabach, T. Steinbach, S. Stender, G. F. Viamontes, and J. Mohnke. ProtoMol: An object oriented framework for molecular dynamics. <http://www.cse.nd.edu/~1c1s>, 2000.
41. L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten. NAMD2: Greater scalability for parallel molecular dynamics. *J. Comp. Phys.*, 151:283–312, 1999.
42. M. Karplus. Molecular dynamics: Applications to proteins. In J.-L. Rivail, editor, *Modelling of Molecular Structures and Properties*, volume 71 of *Studies in Physical and Theoretical Chemistry*, pages 427–461, Amsterdam, 1990. Elsevier Science Publishers. Proceedings of an International Meeting.
43. 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. *Chem. Phys. Lett.*, 313:261–266, 1999.
44. S. Kuwajima and A. Warshel. The extended Ewald method – A general treatment of long-range electrostatic interactions in microscopic simulations. *J. Chem. Phys.*, 89:3751–3759, 1988.
45. A. R. Leach. *Molecular Modelling, Principles and Applications*. Addison Wesley Longman Limited, Essex, 1996.
46. A. D. MacKerell Jr., D. Bashford, M. Bellott, R. L. Dunbrack Jr., J. Evanseck, M. J. Field, S. Fischer, J. Gao, H. Guo, S. Ha, D. Joseph, L. Kuchnir, K. Kuczera, F. T. K. Lau, C. Mattos, S. Michnick, T. Ngo, D. T. Nguyen, B. Prodhom, I. W. E. Reiher, B. Roux, M. Schlenkrich, J. Smith, R. Stote, J. Straub, M. Watanabe, J. Wiorkiewicz-Kuczera, D. Yin, and M. Karplus. All-hydrogen empirical potential for molecular modeling and dynamics studies of proteins using the CHARMM22 force field. *J. Phys. Chem. B*, 102:3586–3616, 1998.
47. T. Matthey and J. P. Hansen. Evaluation of MPI's one-sided communication mechanism for short-range molecular dynamics on the Origin2000. In *PARA2000 and Workshop on Applied Parallel Computing*, 2000.
48. T. Matthey and J. A. Izaguirre. 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, March 2001. Society for Industrial and Applied Mathematics.
49. T. M. Nyman and P. Linse. Ewald summation and reaction field methods for potentials with atomic charges, dipoles, and polarizabilities. *J. Chem. Phys.*, 112:6152–6160, 2000.
50. S. Reich. Dynamical systems, numerical integration, and exponentially small estimates, 1998. Habilitation Thesis.

51. J.-P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen. Numerical integration of the Cartesian equations of motion of a system with constraints: Molecular dynamics of  $n$ -alkanes. *J. Comp. Phys.*, 23:327–341, 1977.
52. A. Sandu and T. Schlick. Masking resonance artifacts in force-splitting methods for biomolecular simulations by extrapolative Langevin dynamics. *J. Comput. Phys.*, 151(1):74–113, May 1, 1999.
53. J. Sanz-Serna and M. Calvo. *Numerical Hamiltonian Problems*. Chapman and Hall, London, 1994.
54. T. Schlick. Some failures and successes of long-timestep approaches to biomolecular simulations. In P. Deuffhard, J. Hermans, B. Leimkuhler, A. Mark, S. Reich, and R. D. Skeel, editors, *Algorithms for Macromolecular Modelling*, volume 4 of *Lecture Notes in Computational Science and Engineering*, pages 221–250. Springer-Verlag, 1998.
55. T. Schlick, M. Mandziuk, R. D. Skeel, and K. Srinivas. Nonlinear resonance artifacts in molecular dynamics simulations. *J. Comput. Phys.*, 139:1–29, 1998.
56. K. E. Schmidt and M. A. Lee. Multipole Ewald sums for the fast multipole method. *J. Stat. Phys.*, 89:411–424, 1997.
57. SGI. The Standard Template Library: Introduction. [http://www.sgi.com/Technology/STL/stl\\_introduction.html](http://www.sgi.com/Technology/STL/stl_introduction.html).
58. 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*, 1998. Also available from [http://www.lsc.nd.edu/downloads/research/mtl/papers/mtl\\_poosc.pdf](http://www.lsc.nd.edu/downloads/research/mtl/papers/mtl_poosc.pdf).
59. R. D. Skeel. Macromolecular dynamics on a shared-memory multiprocessor. *J. Comp. Chem.*, 12(2):175–179, January 1991.
60. R. D. Skeel. Integration schemes for molecular dynamics and related applications. In M. Ainsworth, J. Levesley, and M. Marletta, editors, *The Graduate Student's Guide to Numerical Analysis*, SSCM, pages 119–176. Springer-Verlag, Berlin, 1999.
61. R. D. Skeel and J. Izaguirre. The five femtosecond time step barrier. In P. Deuffhard, 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 303–318. Springer-Verlag, Berlin Heidelberg New York, Nov. 1998.
62. R. D. Skeel, G. Zhang, and T. Schlick. A family of symplectic integrators: stability, accuracy, and molecular dynamics applications. *SIAM J. Sci. Comput.*, 18(1):203–222, Jan. 1997.
63. P. E. Smith and B. M. Pettitt. Efficient Ewald electrostatic calculations for large systems. *CPC*, 91(1):339–344, September 1995.
64. D. Solvason, J. Kolafa, H. G. Petersen, and J. W. Perram. A rigorous comparison of the Ewald method and the fast multipole method in 2 dimensions. *Comp. Phys. Comm.*, 87:307–318, 1995.
65. J. Stone, J. Gullingsrud, and K. Schulten. A system for interactive molecular dynamics. In *2001 Symposium on Interactive 3D Graphics*. ACM, 2001. In Press.
66. B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, third edition, 1997.
67. M. Tuckerman, B. J. Berne, and G. J. Martyna. Reversible multiple time scale molecular dynamics. *J. Chem. Phys.*, 97(3):1990–2001, 1992.

68. M. E. Tuckerman, D. Yarne, S. O. Samuelson, A. L. Hughes, and G. J. Martyna. Exploiting multiple levels of parallelism in Molecular Dynamics based calculations via modern techniques and software paradigms on distributed memory computers. *CPC*, 128:333–376, 2000.
69. T. Veldhuizen. Blitz++: The library that thinks it is a compiler. Conference presentation, Extreme! Computing Laboratory, Indiana University Computer Science Department, Sep. 1998. <http://oonumerics.org/blitz/blitztalk.ps.gz>.
70. T. Veldhuizen. Techniques for scientific c++. Technical Report 542, Indiana University Computer Science Department, 2000. <http://extreme.indiana.edu/~tveldhui/papers/techniques/>.
71. J. Vincent and K. M. Merz. A highly portable parallel implementation of Amber using the message passing interface standard. *J. Comp. Chem.*, 11:1420–1427, 1995.