DOI 10.1002/jcc.21915

DynamO: A Free $\mathcal{O}(N)$ General Event-Driven Molecular Dynamics Simulator

M. N. Bannerman,*[a,b] R. Sargant,[b] and L. Lue[c,d]

Molecular dynamics algorithms for systems of particles interacting through discrete or "hard" potentials are fundamentally different to the methods for continuous or "soft" potential systems. Although many software packages have been developed for continuous potential systems, software for discrete potential systems based on event-driven algorithms are relatively scarce and specialized. We present DynamO, a general event-driven simulation package, which displays the optimal $\mathcal{O}(N)$ asymptotic scaling of the computational cost with the number of particles N, rather than the $\mathcal{O}(N \log N)$ scaling found in most standard algorithms. DynamO provides reference implementations of the best available event-driven algorithms. These techniques allow the rapid simulation of both complex and large ($>10^6$ particles) systems for long times. The performance of the program is benchmarked for elastic hard sphere systems, homogeneous cooling and sheared inelastic hard spheres, and equilibrium Lennard–Jones fluids. This software and its documentation are distributed under the GNU General Public license and can be freely downloaded from http://marcusbannerman.co.uk/dynamo. © 2011 Wiley Periodicals, Inc. J Comput Chem 32: 3329–3338, 2011

Keywords: molecular dynamics • event-driven simulation • discontinuous potentials • hard spheres • square-well potential

Introduction

Molecular dynamics (MD) simulations have become an indispensable tool in the development of novel nanomaterials,[1] drug discovery,[2–4] and materials engineering in the estimation of thermophysical properties and phase behavior of complex solutions. MD not only allows the exploration of the link between interparticle interactions and macroscopic structure and dynamics but also is capable of providing quantitative predictions for real materials. MD simulations have been dominated by time-stepping methods for systems that interact with continuous potentials. This method was first used by Rahman[5] in 1964 and later popularized by Verlet.[6] Since then, many sophisticated software packages have been developed, such as large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS),[7] Groningen Machine for Chemical Simulation (GROMACS),[8,9] NAMD,[10] Desmond,[11] and Extensible Simulation Package for Research on Soft Matter (ESPResSo),[12] which are freely available and allow the simulation of complex systems. In addition, a wealth of force fields based on continuous potentials have been developed to describe real materials, such as small organic and inorganic molecules, polypeptides, proteins, and DNA (e.g., see AMBER[13,14] or CHARMM[15]). MD has also been applied to granular systems, which was pioneered by Cundall and Strack.[16] Since then, Hertz's law for elastic particles has been generalized for viscoelastic spheres,[17] and a range of approximations for the tangential forces are now available.[18–20]

An alternative approach to modeling many-body systems is through the use of discrete interaction potentials, such as the hard-sphere or square-well potentials. These potentials contain only distinct energy level changes; however, they can be stepped to either approximate soft potentials, such as the Lennard–Jones potential,[21] or directly reproduce thermodynamic data.[22]

The "true" interactions between real molecules or atoms (or larger scale particles) are expected to be smooth and continuous, and so one may question the relevance of discrete potentials; they are an extreme approximation to the "true" interactions. However, in reality, all interaction potentials that are used in computer simulations are necessarily approximate, due to practical limitations in computing resources. In the case of continuous potentials, the main approximation is typically the assumption of pair-wise additivity, neglecting many-body interactions, which are present in all "real" systems, or the use of restricted functional forms for the interaction potential (e.g., Lennard–Jones, Stockmayer, etc.). So most commonly used continuous potentials also only approximate the "true" interactions.

The main issue is, however, not whether a potential exactly reproduces the "true" interactions between real particles, but whether it captures the essential features of the interaction to be able to reproduce the physics/chemistry which is of interest in a particular study. This is the primary motivation of coarse grained simulations, and many coarse-grained potentials have also been

[a] *M. N. Bannerman*
Institute for Multiscale Simulation, Universität Erlangen-Nürnberg, Erlangen, Germany
E-mail: marcus.bannerman@cbi.uni-erlangen.de

[b] *M. N. Bannerman, R. Sargant*
School of Chemical Engineering and Analytical Science, The University of Manchester, Oxford Road, Manchester M13 9PL, United Kingdom

[c,d] *L. Lue*
Department of Chemical and Process Engineering, University of Strathclyde, James Weir Building, 75 Montrose Street, Glasgow G1 1XJ, United Kingdom

Contract/grant sponsor: EPSRCDIA

developed to describe systems on larger scales, such as the M3B model for carbohydrates[23] and the MARTINI force field, which was originally developed for lipids[24] and later extended to proteins.[25] It is within this context that discrete potentials are extremely useful.

Although simulations for discrete potential systems are not as prevalent as for continuous potential systems, the literature for classical systems of particles with discontinuous potentials is quite extensive. Indeed, Alder et al.[26] reported MD simulations for systems of hard spheres using an event-driven algorithm in 1957, 7 years before Rahman's soft potential simulations. Since then, many discrete potential force fields have been developed for a wide range of systems, including granular materials,[27] simple molecular systems (e.g., SPEADMD[28]) such as mixtures of hydrocarbons,[28] ethers,[28] alcohols,[29] amines,[29] and carboxylic acids,[30] block copolymer micelles[31] and organized mesophases,[32] and detailed models (e.g., PRIME[33]) for polypeptide[34,35] and protein[36] solutions. These models not only offer qualitative insight to these systems but also provide quantitative predictions for such properties as vapor–liquid phase equilibrium.[28] In addition, the models are detailed enough to realistically capture protein structure, but sufficiently efficient to examine folding and aggregation.[33,36]

In coarse-grained simulations, discrete potentials possess an advantage over continuous potentials in their greater simplicity. The methods used to simulate discrete potential systems offer great computational advantages at low to moderate densities and allow computational resources to be focused on the regions in space and time where relevant dynamics occurs. Consequently, much larger length and time scales can be explored for discontinuous coarse-grained potentials than equivalent coarse-grained models based on continuous potentials. A comparison of relative advantages of discrete and continuous potential models, within the context of fibril formation in polypeptides, is provided in Ref. [34].

Standard numerical methods developed for integrating systems interacting through soft potentials are inefficient for systems with discrete potentials, due to discontinuities in the potential. A time-stepping algorithm, where changes in the interaction energies are detected after the time step is taken, is still feasible[37]; however, this method is necessarily approximate, and high accuracy requires a small, computationally expensive time-step. Event-driven algorithms avoid these difficulties by detecting the time of the next interaction *a priori*. The system is then analytically integrated to the time of the next interaction (event) in a single step. In principle, event-driven algorithms provide an exact method for performing MD simulations for discrete potential systems, where the dynamics can be decomposed into a sequence of events. Event-driven MD is extremely efficient for simulating systems of particles interacting through steep potentials with a relatively small number of steps.

There are numerous reviews on algorithms for event-driven molecular dynamics[37–43]; however, it is difficult to find documented implementations of algorithms, which include the source code. Modern algorithms are complex and contain many subtle difficulties which, if poorly implemented, can severely restrict the generality and speed of the code. In this article, we present Dynamics of discrete Objects (DynamO), a free source MD package that is optimized for event-driven dynamics. DynamO is capable of simulating large ($\gtrsim 10^6$ particles) and complex systems for extremely long simulation times.[44] This package has already been used to study sheared/damping granular materials,[45,46] square well molecules,[47] binary mixtures,[48] parallel cubes,[49] and helix forming polymers.[50]

The remainder of this article is organized as follows. In Algorithm Details, we begin with an overview of the basic elements of event-driven MD simulations. This section briefly reviews some of the recent algorithmic advances. Some improvements that we have developed to the simulation algorithm are presented in the Improvements to the Simulation Method section. These include novel approaches to access particle information, optimize neighbor lists, and minimize numerical inaccuracies. The Benchmarking section provides benchmark simulations for DynamO on systems of single component hard spheres and stepped Lennard–Jones molecules. These simulations provide timing results and information on the scaling of the calculation times with system size. Finally, the conclusions of the article are presented, along with a discussion of possible directions for future extensions to DynamO. An outline of the general structure of DynamO and details of various aspects of its implementation, as well as a listing of its currently implemented features, are provided in the Appendix .

Algorithm Details

A MD simulation calculates the trajectory of a collection of a large number of interacting particles. When the particles interact with each other through a discrete potential, the dynamics of the system are governed by a series of distinct events (e.g., collisions between particles). These events may alter the properties of the particles, such as their velocities. Between these events, the particles move on a ballistic trajectory, and the dynamics of the system is known analytically. This is a significant advantage of event-driven algorithms, as the ballistic motion requires no numerical integration and does not suffer from truncation error.

In an event-driven simulation, there are three major tasks, which occupy most of the computational time: (i) searching for events (event detection), (ii) maintenance of the event list, and (iii) execution of events. An outline of a basic event-driven simulation algorithm is given below, including the scaling of the computational cost of each step with the number of particles N in the system:

1. Event testing $\mathcal{O}(N^2)$: All particles and pairs of particles are tested to determine if/when the next interaction occurs. The times of these events are inserted into the future event list (FEL).

2. Event sorting $\mathcal{O}(N)$: The events in the FEL are sorted to determine the next event to occur.

3. Motion of the system $\mathcal{O}(N)$: The system is evolved, or free streamed, to the time of the next event.

4. Execution of the event $\mathcal{O}(1)$: Particles involved in the event are updated with new velocities.

5. Update events in FEL $\mathcal{O}(N)$: Events in the FEL that involve particles that have just undergone the executed event are now invalid. New events for these particles are possible and must be tested for. The future event list may be cleared and rebuilt ($\mathcal{O}(N^2)$), or only the affected events can be updated ($\mathcal{O}(N)$).

6. End condition: Continue to step 2 unless sufficient collisions have been executed or the maximum simulation time has elapsed.

Alder and Wainwright[26] proposed the first algorithm for event-driven MD simulations. Since their pioneering work, there have been significant advances in the development of this initial algorithm. We will briefly cover the advances in event detection/execution before detailing improvements in the maintenance of the list of all possible future events.

Alder and Wainwright[38] were the first to suggest the use of a "neighbor list" technique to improve the efficiency of the search for future events. In this method, the simulation box is divided into small cells. An additional event type is also introduced to track the motion of the particles between these cells. These cells are used to determine which particles are close to a particle undergoing an event. This significantly reduces the number of particles that need to be tested for new interactions in step 1 to $\mathcal{O}(N)$ and in step 5 to $\mathcal{O}(1)$. This "neighbor list" technique has since been extended to infinite systems[51] using a hashing technique, overlapping cells[52] to reduce the frequency of cell transitions, and hybrid methods[53] combining multiple neighbor lists. All of these methods are available in DynamO.

One of the most significant advances in event-driven simulation came with the development of asynchronous[40,54] or "delayed states"[41] algorithms. In these algorithms, only particles involved in an event or within the neighborhood of an event are updated when an event occurs (step). Each particle stores the time at which it was last updated and is only evolved to the current time when it is tested for, or undergoes an event. As particles that are not involved in an event and in step 5 to $\mathcal{O}(1)$.

In most modern implementations of event-driven algorithms, the system size scaling of the event computational cost arises from the maintenance[39] of a list of all possible future events (step 2). This list needs to be sorted to determine the next event to occur, and after an event is executed the list must be updated. Improvements in sorting began with the use of many variants of binary trees[39] before complete binary trees were found to be optimal.[55] Complete binary trees exhibit an $\mathcal{O}(\log_2 N_{\text{event}})$ scaling in the number of events N_{event} contained in the tree. It has been previously suggested[39,40] that an $\mathcal{O}(\log_2 N_{\text{event}})$ scaling of sorting the event list is the asymptotic minimum; however, a recent advance in event sorting using calendar event queues[56] now enables $\mathcal{O}(1)$ scaling. This is achieved by first presorting events into fixed intervals of time or "dates" within a "calender." The date to which an event corresponds can be determined by simply dividing the event time by the duration of a date and truncating to an integer ($\mathcal{O}(1)$). As the simulation progresses to a new date in the calendar, the events within a date are sorted and processed using a complete binary tree. The length and duration of the calendar is scaled with the system size, resulting in a fixed average size of this complete binary tree. This ensures that the deletion, insertion, and updating operations on the FEL all remain of order $\mathcal{O}(1)$.

With these methods, the overall computational cost of executing a single event is now independent of the system size, which is the theoretical optimum. The number of events per particle is typically proportional to the total time simulated, thus the computational cost of simulating a unit of time scales as $\mathcal{O}(N)$ with the system size.

Other algorithmic improvements, which do not affect the system size scaling of the computational cost, have been made to the deletion of events from the calendar. The local minima algorithm[41] relies on each event being associated with at least one particle. A priority queue called a particle event list (PEL) is used to sort the events associated with a single particle. The PEL is then inserted into the global event list and sorted according to its earliest event. When a particle undergoes an event, at least half of the invalidated events associated with it can be deleted by simply erasing the corresponding PEL. The remainder of the invalidated events are left in the FEL and are deleted if they reach the top of the FEL. This is achieved by tracking the number of events each particle has undergone in total and the value of this when the event was tested.[41]

Improvements to the Simulation Method

In this section, we present some additional improvements to the event-driven simulation algorithm that we have developed and implemented within DynamO. These methods are primarily concerned with the storage of data and increasing simulation accuracy.

Particle data

The main bottleneck in most scientific programs is the speed of memory accesses, and event driven simulation is no exception. It is typically cheaper to calculate values than to store them in memory. In DynamO, this approach is applied to all static values or "properties." Associated to each particle is a class, which contains only a particle's position, velocity, ID number, and the time, the particle was last updated. All other properties, dependent on the system studied (e.g., mass/inertia, orientation, species), are accessed using the particle's ID number when required.

A design feature of DynamO is the use of functions, as opposed to look-up tables, to perform this look-up both when mapping a property (such as mass) to a range of particles and when determining the values of a property. For example, in a single component system, all particles have identical masses. To avoid storing redundant information, a distribution representing a single value of the mass is stored behind a function that maps it to all particles. These functions behave like a standard STL-container and are used to define molecular topology, species, mass, and how particles interact (e.g., to create mixtures of particles). This approach conserves memory, as an absolute minimal number of entries are required,

and is faster than a look-up table due to the reduced number of memory accesses.

For dynamical properties of particles where this approach is impossible (e.g., if two particles have captured each other in an attractive well), DynamO makes use of unordered sets and maps. These hashed containers still provide $\mathcal{O}(1)$ operations when using a suitable hash function but conserve memory when compared to using arrays.

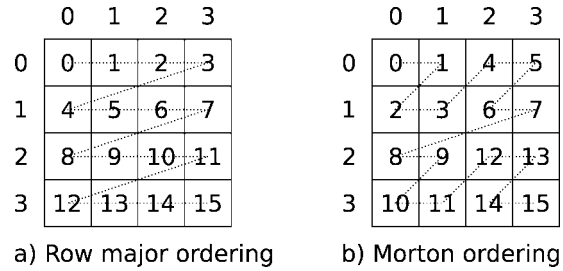

Figure 1. A two dimensional 4×4 array stored in row major order and Morton order in memory.

Morton ordered neighbor lists

The algorithms discussed in Algorithm Details section result in a theoretical system size scaling of $\mathcal{O}(1)$ in the cost of processing a single event. In practice, the computational cost is affected by the memory architecture of the computer that runs the simulation. If the events are relatively inexpensive to test for, the bulk of the simulation time will be spent on memory accesses to retrieve particle, event, and neighbor list data.

A fundamental aspect of modern processors is the use of a CPU cache, into which data are "fetched" before becoming available to the running process. A cache "miss" occurs when data to be accessed are not already available in the cache. The cost of a cache miss is typically quite severe, requiring several computational cycles to fetch the data from main memory and load it into the CPU cache. Data are typically fetched in blocks of 64 bytes; therefore, data localized in memory are typically fetched at the same time. Thus, if the location of data in memory is strongly correlated to the data access pattern, then the number of cache misses can be reduced.

The particle and event data accesses are effectively random, which renders any attempt to optimize the access patterns futile; however, accesses to the neighbor list data are strongly correlated. Whenever a particle's local space is to be inspected for possible events, the cells of the neighbor list, which surround the particle are checked for event partners. In the standard implementation of a neighbor list,[57] only the first element of each cell is correlated this way as a singly linked list is used look-up all other contained particles. Nevertheless, accesses to the first particle in the neighbor list cell are strongly correlated in the spatial coordinates of the cell. Thus, if the cell's first particle data are arranged such that spatially localized data are also localized in memory, the number of cache misses will be reduced.

Arrays are typically stored linearly (row major in C++) in computer memory, where each successive row of data in the lowest spatial dimension is appended to the previous row (see Fig. 1a). An alternative space filling curve which retains a high level of spatial locality is the Morton-order or "Z-order" curve (see Fig. 1b). Recently, fast methods for dilating integers used in Morton ordering[58] and methods for directly carrying out mathematical operations on the dilated integers[59] have become available. The overhead of calculating a three dimensional Morton number is now less than the cost of a cache miss in many applications. In DynamO, we have implemented Morton ordering in the neighbor list, and a comparison between linear and Morton order is presented in the timing results section (see Benchmarking section).

The algorithms described thus far are primarily concerned with increasing the speed of the calculations; however, the numerical accuracy of the simulation is crucial. The methods used to ensure precision is maintained are detailed in the following section.

Min-max particle event lists

Priority queues are often used for the PEL as they allow insertion, access to the shortest time event, and clearing; and are optimal for the access patterns of the PEL.[60] However, there are several drawbacks to using these containers, which result from their ability to have a dynamic size: Often only the first few events in the PEL are relevant to the dynamics and yet all tested events are stored in the priority queue. In addition, the methods for manipulating dynamic memory (new and delete in C++) are slow and result in an extra layer of indirection. One method proposed to solve these drawbacks is to only store the earliest event in a particle's event list. If this single stored event is invalidated, all possible events of the particle must be recalculated and the new minimum stored.[40] Unfortunately, general event-driven simulations utilize many types of virtual events (e.g., neighbor list boundary crossings), resulting in many recalculations. Modern event driven potentials, such as those for asymmetric particles, can also be expensive to recompute.

It would be preferable to allow the PEL to only store some small, but greater than one, number of events. This would limit the number of events in the PEL (to save memory) and yet still store a sufficient number to minimize recalculations which occur when the PEL is empty. This fixed-size PEL must provide constant-time access to the earliest event (for the dynamics) and also provide constant-time access to the last event in the list to mark it as a full-recalculation event if an event is added to a filled PEL. This last element may also be used as a early test mechanism to determine if an event needs to be inserted into a filled PEL.

MinMax heaps[61] are a data structure which satisfies all of these requirements. A comparison of MinMax heaps, single event storage, and a standard template library (STL) priority_queue is presented in Figure 2. Even for hard sphere systems, the MinMax PEL offers the speed advantage of a priority_queue with a great saving in memory cost. A slight speed improvement from the priority_queue is expected, as the MinMax PEL does not require dynamic memory. The optimal size of a MinMax queue appears to be three stored events. This optimum is dependent on how many invalid or virtual events are expected to appear at the top of the

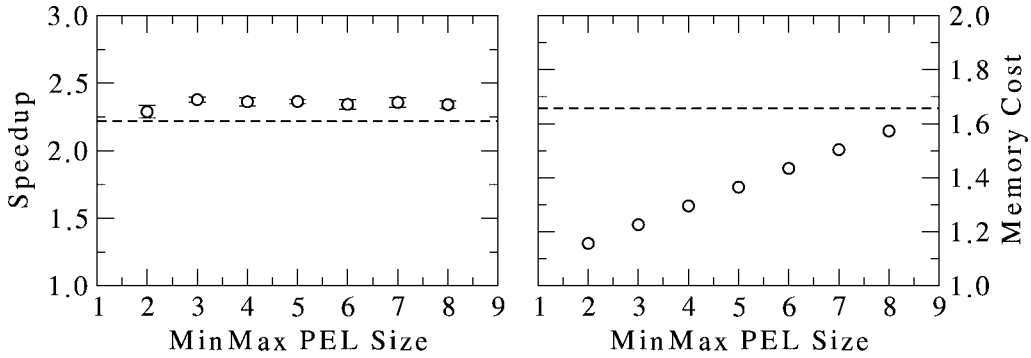

Figure 2. The speedup and memory cost of using a MinMax PEL of various sizes relative to storing only a single event per particle. The dashed lines are the values for using a STL priority_queue. The results are for a monodisperse system of $N = 10^5$ elastic hard spheres simulated on an Intel® Core™i5 750 with 8 MB L3 cache and 8 GB of RAM.

PEL; however, a more conservative size of 4 or 5 may be selected without too great an increase in the memory cost. All further simulation results presented in this paper are performed with a MinMax PEL size of 4.

Accuracy and time invariance

MD simulations require a high level of accuracy. This is especially true for "hard core systems," where configurations with overlapping hard cores resulting from numerical inaccuracies are unphysical and impossible to resolve. Accuracy is particularly important in inelastic, granular systems due to clustering of the particles; small errors in the movement of particles, resulting in overlaps, are increasingly probable.

Several improvements have been made to the algorithm used in DynamO to maintain the numerical precision of the simulation. The use of the "delayed states"[41] algorithm already reduces the number of times a particle is free-streamed between events. Also, when an event is scheduled to occur, events are retested to determine the exact time at which it occurs.[60] This reduces the likelihood of overlaps occurring due to inaccuracies in the free streaming. Furthermore, the dynamics of the system is tracked to ensure invalid events cannot occur (e.g., particles must be approaching to be tested for a collision, square-well molecules must have been captured to test for a release event). This implies that the dynamics of the system must always be deterministic; however, random events, such as those that occur in the Andersen thermostat,[62] are possible by randomly assigning a time after each random event and scheduling this fixed time in the event list.

Another inaccuracy arises from the storage of absolute times in the simulation. Events occur at an absolute time t_e and, within the asynchronous algorithms, the particle data are stored at a certain absolute simulation time t_p. As the absolute simulation time t_{sim} increases in magnitude, round-off error will accumulate in these stored absolute times. To avoid this, only time differences $\Delta t_{e/p}$, recording the time relative to the current absolute simulation time $t_{e/p} = t_{sim} + \Delta t_{e/p}$, must be stored. This does, however, introduce an $\mathcal{O}(N)$ computational cost per event in maintaining these time differences. This is alleviated by storing time differences relative to a reference time difference $\Delta t_{e/p} = \Delta t_{ref} + \Delta t_{e/p}^{(stored)}$. This reference time difference Δt_{ref} is updated at every event without incurring significant cost. Periodically all time differences and reference time differences must be synchronized to prevent round-off error, and the interval at which this synchronization occurs is proportional to the system size. This leads to a time invariant simulation algorithm with a fixed upper bound on round-off error without affecting the scaling of the computational cost.

Visualization

DynamO is capable of simulating millions of particles at close to real time. With macroscopic simulations of granular systems, the simulation speed approaches the timescale of the interesting dynamics. To enable live visualization of these massive systems and to allow interactive or "steered" simulations, a new visualizer was written in OpenCL/GL. This library, known as Coil, is capable of rendering up to a million spheres in real time with full diffusive, specular and shadow lighting calculations, and HDR effects. The result is publication-quality images (see Fig. 3) at interactive frame-rates while only using a single core of the host CPU. This visualization library is already finding application in a wide range of simulators outside of DynamO.

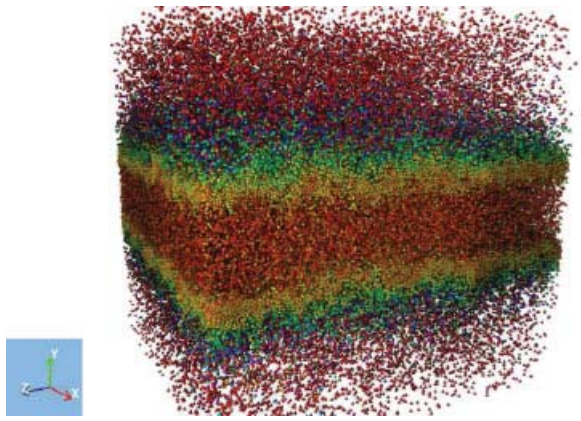

Figure 3. A snapshot of 5×10^5 sheared inelastic hard spheres, generated live from a simulation, displaying a characteristic clustering behavior of granular systems. On a NVIDIA® Geforce GTX 260™, the image is updated at 15 frames per second.

Summary

In this section, we have discussed some of the new methods we have developed to improve the computational efficiency of DynamO. The next section provides some benchmarking simulations for DynamO and tests the system size scaling of the simulation code.

Benchmarking

In order to benchmark the speed of DynamO and test that the optimal $\mathcal{O}(1)$ scaling is achieved, we perform MD simulations on systems composed of hard spheres. This interaction is relatively inexpensive and as such is useful in testing the performance of the simulation framework. Each sphere has a diameter σ and is run over a range of reduced number densities $\rho\sigma^3$. The simulations were performed on a desktop computer with an Intel®Core™i5 750 processor with 8 GB of RAM. The simulations utilized only a single core of the processor and are averaged over 4 runs of 5×10^6 collisions. The optimal parameters for the bounded priority queue are determined at the start of the simulation by instrumenting the initial event distribution. N calendar dates are used with a width equal to the mean time between events.

The average number of collisions per second is plotted in Figure 4a as a function of system size and density. It is apparent that the memory architecture plays a large role in the speed of the simulation.[56] The rate of collision maintains a relatively constant value when the program fits inside the CPU cache ($\lesssim$8 MB boundary, $N \lesssim 1.6 \times 10^4$), and for very large systems ($N \gtrsim 10^5$) where the cache effects are proportionally small. Accounting for these memory size effects, the algorithm appears to exhibit $\mathcal{O}(1)$ scaling of the collision cost. Inside the cache, the simulation reaches a maximum of roughly 2.3×10^5 events per second, compared to a minimum of $\sim 7 \times 10^4$ events per second outside of the cache. Beyond the 8-GB memory limit ($N \approx 1.6 \times 10^7$), disk swapping begins to occur and the performance is substantially degraded. The event processing rate is relatively insensitive to density. A slight decrease in the event processing rate is expected at higher densities as local neighbor lists contain more entries (neighbors).

A comparison between Morton ordering and the typical linear ordering is presented in Figure 5. Even inside the 8-MB cache

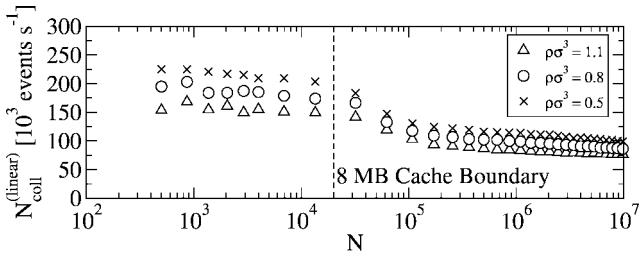

Figure 4. The number of collision events processed per second using linear neighbor lists $N_{coll}^{(linear)}$ as a function of (a) the number of particles N and (b) the number density ρ. The lines indicating the cache memory boundary is approximate as different densities incur slightly different memory requirements. The results are for a monodisperse system of elastic hard spheres simulated on an Intel®Core™i5 750 with 8-MB L3 cache and 8 GB of RAM.

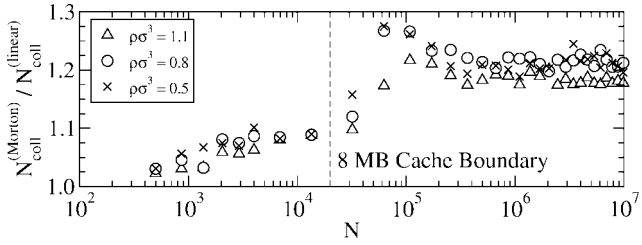

Figure 5. The number of events processed per second using linear and Morton ordered lists, $N_{coll}^{(linear)}$ and $N_{coll}^{(Morton)}$, respectively. The dashed lines and system are described in Figure 4.

limit, the Morton ordering has a positive effect. This may be due to localisation of memory accesses in the smaller L1 cache. For more practical system sizes operating outside the cache, Morton ordering offers a 10–28% increase in event processing speed. This is remarkable as only a single unsigned integer per cell is actually optimized using this technique, proving its utility even in event-driven simulations. At higher densities, the Morton ordering has a slightly reduced effect as the ratio of particle to neighbor list memory accesses is increased. Overall, Morton ordering appears to be an effective method of increasing the computational speed by reducing the number of cache misses within a simulation.

The large effect of caching on the performance of the simulation indicates at least half the time of simulations outside the cache boundary are spent waiting on memory accesses. Cache simulations have been performed using Callgrind,[63] and for a density of $\rho\sigma^3 = 0.5$, approximately half of the cache misses result from accesses to the contents of the neighbor lists. The remaining cache misses are associated with accesses to particle and event data.

The code is also benchmarked for some more complex systems. The collision model is extended to inelastic hard spheres with elasticity e. In cooling simulations, the system is bounded by standard periodic boundary conditions, and the temperature is rescaled to unity every 2×10^6 collisions. For sheared systems, Lees-Edwards boundary conditions are used and rescaling is not required. Inelastic particles ($e < 1$) tend to cluster (see Fig. 3) which increases the cost of updating the event list after a collision (see Fig. 6a). However, at very low inelasticities, the event processing rate increases as events become correlated to "rattling" particles, improving the cache's effectiveness and reducing the effect of Morton-ordered neighbor-lists. This behavior is also apparent in the sheared inelastic simulations (see Fig. 6b). Morton ordering is quite effective in inelastic simulations, with a slight enhancement over elastic

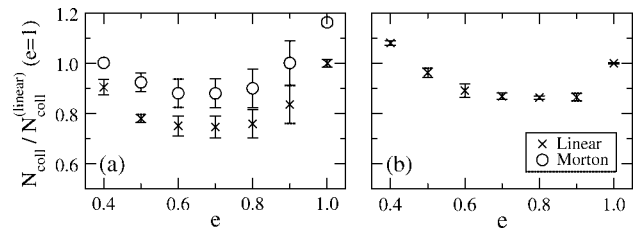

Figure 6. The speedup factor versus density for (a) cooling and (b) sheared inelastic systems with $N = 10^5$. The speedup factor is relative to the linear-neighbor-list elastic system.

simulations due to the increased spatial correlations in the system. Morton ordering could not be tested in the sheared system as it has not yet been ported to the sheared neighbor-list.

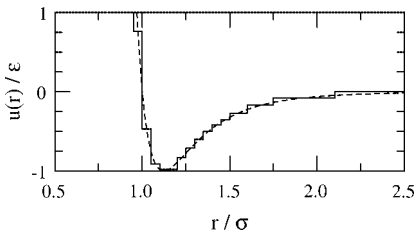

Figure 7. A stepped potential (solid line) that approximates the Lennard-Jones potential (dashed line) (potential 6 of Ref. [21]).

Finally, in order to assess the relative performance of an event-driven algorithm with a time-stepping algorithm, simulations are performed for the stepped and continuous variants of the Lennard–Jones potential[21] (see Fig. 7). This potential provides a reasonable approximation to the Lennard–Jones fluid and demonstrates the applicability of discrete potentials to simple fluids. The continuous potential is simulated using GROMACS 4.5.4,[9] a popular and highly optimized time-stepping molecular dynamics package. This comparison is biased in favor of the time-stepping algorithm due to the shape of the potential and the relative maturity of the GROMACS code; however, it should be noted that, unlike time-stepping MD, event-driven algorithms do not use a numerical integration scheme and are accurate to the numerical precision of the machine. Both simulations consist of $N = 13500$ Lennard–Jones atoms with mass m, run for a simulation length of $t = 50(m\sigma^2/\epsilon)^{1/2}$, and using double precision calculations. The GROMACS simulations used the velocity Verlet integrator, Verlet lists, a reduced time step of $\Delta t = 0.005(m\sigma^2/\epsilon)^{1/2}$, and a reduced cutoff distance of 3σ.

A comparison of the calculated radial distribution functions and the relative speed of the simulators are presented in Figure 8. The radial distribution functions are in close agreement, with a slight underestimation in the DynamO results at a distance of $r/\sigma \approx 1.1$. At low densities, DynamO significantly outperforms GROMACS as expected as event-driven dynamics is optimal in collisional regimes. Surprisingly, DynamO also performs well into the liquid phase, with GROMACS only displaying a $4\times$ speedup. This is a small cost when it is considered that the event-driven algorithm solves its dynamics without truncation error.

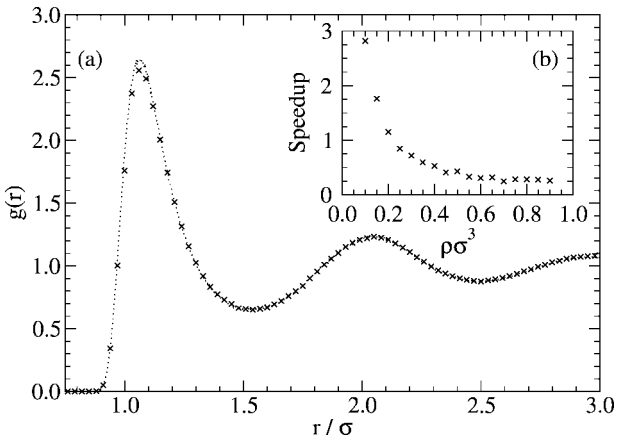

Figure 8. A comparison of (a) the radial distribution function $g(r)$ of a Lennard–Jones fluid at a number density of $\rho\sigma^3 = 0.85$ and a temperature of $k_BT/\varepsilon = 1.3$ calculated using GROMACS (dotted line) and using DynamO (crosses) and (b) the speed of the DynamO simulations relative to the GROMACS simulations at $k_BT/\varepsilon = 1.3$ over a range of densities.

Conclusions

We have detailed the fundamental components of DynamO, a modern event-driven MD simulation package. The program is distributed under the GNU General Public License. The full source code and documentation are freely available online at http://www.marcusbannerman.co.uk/dynamo. The program provides reference implements for many modern algorithms for event-driven simulations and also includes several new techniques for mitigating round-off error, improving speed, and optimizing memory access patterns. The latter is achieved by preserve cache locality through using Morton ordering to store neighbor entries in spatially localized clusters. The speed of accessing memory appears to be a significant bottleneck in simulating systems with simple potentials. Through benchmark simulations on single component elastic-hard-sphere systems, we have demonstrated that DynamO exhibits an $\mathcal{O}(1)$ scaling with system size of the computational cost of executing events. This leads to an overall scaling of $\mathcal{O}(N)$ for a set duration of simulation time. This allows the rapid simulation of both complex and large (10^7 particle/atom) systems while extracting the long-time behavior.

Many systems can be explored with the package in its current state; however, there are a few planned extensions which will bring the package to the level of generality of modern soft potential packages.

Stepped potentials[21] are already available in DynamO, allowing the straightforward approximation of rotationally symmetric soft potentials. Asymmetric potential dynamics are significantly more complex, especially in the case of hard particles.[64] The determination of the time to collision requires considerable care, and these algorithms are often specialized to the underlying potential.[65] On the other hand, soft potential dynamics are widely used for modelling due to the relative ease with which new potentials can be implemented. A natural step forward for event-driven dynamics is in the implementation of the framework developed by van Zon and Schofield,[66] which generalizes the implementation of asymmetric discrete models by using a soft potential to generate and solve the dynamics of an equivalent "terraced" potential. A partial implementation is already available, although care must still be taken in the discretization of the soft-potential; further research is needed to develop this technique.

Long-ranged potentials, such as those due to electrostatic interactions, do not yet have an event-driven equivalent. The implementation of a stepped force field is not difficult; however, coupling the particle positions to the field is not trivial.[42] The detection of events becomes prohibitively expensive due to the added complexity of the free flight phase and typically only exists to ensure that an underlying time stepping integration does

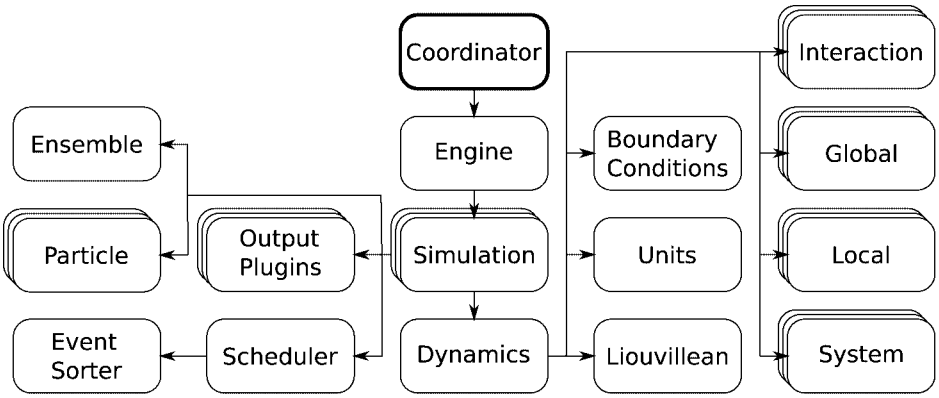

Figure 9. The class hierarchy of DynamO. Only the classes key to the algorithm are displayed. Arrows indicate the nesting of classes, and stacked boxes indicate multiple instances of the class may occur.

not fail by incorrectly generating overlapping hard particles.[42] In the limit of a weak and long-ranged field within a large system, this coupling might be implemented as a boundary condition of a stepped potential grid. These techniques must be developed before event-driven dynamics can be effectively utilized in modeling charged systems.

Event-driven simulations are serial by nature; however, attempts have been made to develop parallel algorithms.[67,68] The simulations are split into computational cells and divided among a collection of processors. Each cell is then run independently of all others until an event occurs at the boundaries of the cells, forcing a synchronization. As the number of processors increases, the synchronization events become more frequent and will limit the scalability; however, excellent performance has been demonstrated for up to 128 processors.[68] At the core of these parallel algorithms is a serial implementation, and DynamO has already been used to simulate systems of 32 million particles on a single processor. Yet, parallel computation will be required as the complexity of the underlying potentials increases. Further developments are needed to optimize memory usage and to explore the possibility of parallelizing the algorithm within a computational cell.

Appendix

Program Design and Features

The development of DynamO has focused on generating a flexible, modular simulator where systems can be constructed from an array of available interactions, conditions, and dynamics. DynamO is written in C++ using an object orientated design. This helps ensure that the code is both extensible and maintainable, provided the classes have well defined interfaces and tasks. All input and output files are in XML to allow easy generation and alteration of system conditions. The implementation of DynamO utilizes only free, open source libraries, including the BOOST (www.boost.org) libraries.

DynamO was originally written to perform NVE MD simulations of particles interacting through spherically symmetric, discrete potentials. However, because of its flexible design, DynamO has been extended to perform a wide variety of calculations for several different types of systems. The dynamics of infinitely thin lines[65] has already been incorporated within DynamO, and other shapes can also be included. Constant temperature simulations are performed through the use of the Andersen thermostat.[62] Multiple simulations can be executed concurrently and combined with the replica exchange method[69] to expedite the equilibration of systems with rough energy landscapes. Umbrella potentials can also be applied to sample specific regions of the phase space of a system. Finally, stepped potentials can be used to approximate rotationally symmetric soft potentials.

Alternate dynamics can also be easily implemented within DynamO. For example, compression dynamics, where the particles in the system grow with time, is already implemented. DynamO is also used[45,48] as a framework to perform direct simulation Monte Carlo calculations for the Enskog equation.

Simple code hierarchies have been suggested previously[70]; however, the level of complexity of modern simulations requires a finer grained class structure than previously outlined.

The class hierarchy of DynamO is presented in Figure 9. Typically each class has several implementations which are selected at run time through the input files. Below, we detail the scope of each class, along with the currently implemented features:

Coordinator: *Abstracts the user interface and system calls.*

This class encapsulates the entire program and provides the user interface. The initialization of operating system features, such as threading, is also performed here.

Engine: *Organizes a collection of simulations to achieve a task.*

In its simplest form, a single simulation is run to obtain output; however, replica exchange techniques[69] and a method to perform isotropic compression of configurations[71] are also available. The replica exchange technique runs several simulations in parallel with a Monte Carlo move to increase the ergodicity of low temperature trajectories.

Simulation: *Encapsulates a single simulation.*

This class represents a single simulation, containing an array of particles, classes describing the dynamics, and data collection classes. The primary function of the simulation class is to initialize and maintain these classes in evolving the system through time.

Scheduler: *Maintains the list of future events.*

This class is responsible for executing events and maintaining the FEL. Several variants exist, a "dumb" scheduler, a scheduler capable of interfacing with a specialized Global that implements a neighbor list, and a multithreaded neighbor-list scheduler.

Event Sorter: *Sorts events in the FEL.*

Provides a method of sorting the FEL, which contains an array of PELs. A complete binary tree[55] and a bounded priority queue[56] are implemented.

Particle: *Container for single particle data.*

This class encapsulates the minimal single particle data. This includes the particle ID number, position, velocity, and local time.

Output Plugins: *Data collection routines.*

A wide range of plugins are available, including radial distribution functions, the complete set of Green-Kubo expressions for mixtures, and visualization plugins for VMD (www.ks.uiuc.edu/Research/vmd), Povray (www.povray.org), and Geomview (www.geomview.org).

Ensemble: *Describes the ensemble of the simulation.*

This is used to ensure the simulation ensemble is valid for certain output plugins and for replica exchange.

Dynamics: *Encapsulates the dynamics methods of the system.*

The dynamics class initializes and maintains classes relating to the dynamics of the system. The actual dynamics are implemented in classes contained within this class.

Units: *Provides functions to scale between simulation and input units.*

Simulations are typically optimal in a unit set other than the input settings. For example, if the dimensions of the simulation box are scaled to one then the enforcement of the periodic boundary conditions reduces to a rounding operation.[72]

Liouvillean: *Contains simple functions to describe the evolution of the system.*

This class implements event testing for basic shapes (e.g., spheres, lines, planes), particle evolution and event dynamics. These are then used by interactions, locals, and globals to implement an event. Several implementations exist, including Newtonian, isotropic compression, Enskog direct simulation Monte Carlo (DSMC) in SIlod coordinates,[73] and axisymmetric rotational dynamics.

Boundary Conditions: *Specifies the limits of the simulation box.*

Square, rectangular, periodic, sliding brick shearing,[74] and infinite boundary conditions are currently implemented.

Interaction: *Two particle events.*

Interactions between two particles are derived from this class. Many interactions have already been implemented (e.g., stepped potentials, hard spheres, parallel cubes, square wells, square-well bonds,[75] thin needles,[65] and an optimized square-well sequenced-polymer interaction).

Global: *Single particle events.*

Events, which affect only one particle, regardless of its position in the simulation, are derived from this class. This is primarily used for neighbor lists, specializations include overlapping cells,[52] mixed methods,[53] cells in shearing, Morton-ordered neighbor lists, and a sentinel event to ensure the nearest image condition does not result in invalid dynamics at low density/small system size.[65]

Local: *Single particle event, localized in space.*

These events only effect a region of space and are inserted into the neighbor lists when available. This is to reduce the number of event tests required. Solid, thermostatted,[60] and oscillating planar/cylindrical/spherical walls are implemented here.

System: *Simulation and multiple-particle events.*

Any type of event, which is not compatible with the definition of a local, global, or interaction, is implemented here. Includes DSMC interactions,[76] Andersen thermostats,[62] and simulation termination conditions. Umbrella potentials are also implemented here as they are many-body interactions.

[1] J. Fish, J Nanoparticle Res 2006, 8, 577.
[2] T. Hansson, C. Oostenbrink, W. van Gunsteren, Curr Opin Struct Biol 2002, 12, 190.
[3] C.F. Wong and J.A. McCammon, Protein simulations: Advances Prot. Chem 2003, 66, 87.
[4] W. L. Jorgensen, Science 2004, 303, 1813.
[5] A. Rahman, Phys Rev 1964, 136, A405.
[6] L. Verlet, Phys Rev 1967, 159, 98.
[7] S. Plimpton, J Comput Phys 1995, 117, 1.
[8] D. V. D. Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, H. J. C. Berendsen, J Comput Chem 2005, 26, 1701.
[9] B. Hess, C. Kutzner, D. Van Der Spoel, E. Lindahl, J Chem Theory Comput 2008, 4, 435.
[10] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, K. Schulten, J Comput Chem 2005, 26, 1781.
[11] K. J. Bowers, E. Chow, H. Xu, R. O. Dror, M. P. Eastwood, B. A. Gregersen, J. L. Klepeis, I. Kolossváry, M. A. Moraes, F. D. Sacerdoti, J. K. Salmon, Y. Shan, and D. E. Shaw, In Proceedings of the ACM/IEEE Conference on Supercomputing (SC06), Tampa, Florida, 2006.
[12] H.-J. Limbach, A. Arnold, B. A. Mann, C. Holm, Comput Phys Commun 2006, 174, 704.
[13] T. E. Cheatham, III, M. Young, Biopolymers 2001, 56, 232.
[14] J. Ponder, D. Case, Adv Prot Chem 2003, 66, 27.
[15] A. MacKerel, Jr., C. Brooks, III, L. Nilsson, B. Roux, Y. Won, M. Karplus, In The Encyclopedia of Computational Chemistry; von R. Schleyer, P., Ed.; Wiley: Chichester, 1998, 1, 271.
[16] P. A. Cundall, O. D. L. Strack, Géotechnique 1979, 29, 47.
[17] N. Brilliantov, F. Spahn, J.-M. Hertzsch, T. Pöschel, Phys Rev E 1996, 53, 5382.
[18] P. Deltour, J. L. Barrat, Journal De Physique I 1997, 7, 137.
[19] P. K. Haff, B. T. Werner, Powder Technol, 1986, 48, 239.
[20] O. R. Walton, R. L. Braun, J Rheol, 1986, 30, 949.
[21] G. Chapela, L. E. Scriven, H. T. Davis, J Chem Phys 1989, 91, 4307.
[22] J. R. Elliott, Fluid Phase Equilibria 2002, 194–197, 161.
[23] V. Molinero, W. A. Goddard, J Phys Chem B 2004, 108, 1414.
[24] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, A. H. de Vries, J Phys Chem B 2007, 111, 7812.
[25] L. Monticelli, S. K. Kandasamy, X. Periole, R. G. Larson, D. P. Tieleman, S.-J. Marrink, J Chem Theory Comput 2008, 4, 819.
[26] B. J. Alder, T. E. Wainwright, J Chem Phys 1957, 27, 1208.
[27] I. Goldhirsch, G. Zanetti, Phys Rev Lett 1993, 70, 1619.
[28] O. Unlu, N. H. Gray, Z. N. Gerek, J. R. Elliott, Ind Eng Chem Res 2004, 43, 1788.
[29] J. R. Elliott, A. Vahid, A. D. Sans, Fluid Phase Equilibr 2007, 256, 4.
[30] A. Vahid, J. R. Elliott, AIChE J 2010, 56, 485.
[31] J. L. Woodhead, C. K. Hall, Langmuir 2010, 26, 15135.
[32] A. J. Schultz, C. K. Hall, J. Genzer, J Chem Phys 2002, 117, 10329.
[33] H. D. Nguyen, C. K. Hall, Biophys J 2004, 87, 4122.

[34] C. K. Hall, V. A. Wagoner, In Amyloid, Prions, and Other Protein Aggregates, Part B; I. Kheterpal, R. Wetzel, Eds., Academic Press, 2006, 412, 338.
[35] A. J. Marchut, C. K. Hall, Proteins: Struct, Funct, Bioinf 2007, 66, 96.
[36] M. Cheon, I. Chang, C. K. Hall, Proteins Struct Funct Bioinf, 2010, 78, 2950.
[37] M. P. Allen, D. Frenkel, J. Talbot, Comput Phys Rep 1989, 9, 302.
[38] B. J. Alder, T. E. Wainwright, J Chem Phys 1959, 31, 459.
[39] D. C. Rapaport, J Comput Phys 1980, 34, 184.
[40] B. D. Lubachevsky, Int J Comput Phys 1991, 94, 255.
[41] M. Marin, D. Risso, P. Cordero, J Comput Phys 1993, 109, 306.
[42] H. Sigurgeirsson, A. Stuart, W.-L. Wan, J Comput Phys 2001, 172, 766.
[43] A. Donev, Simulation 2009, 85, 229.
[44] M. N. Bannerman, L. Lue, L. V. Woodcock, J Chem Phys 2010, 132, 084507.
[45] M. N. Bannerman, T. E. Green, P. Grassia, L. Lue, Phys Rev E 2009, 79, 041308.
[46] M. N. Bannerman, J. E. Kollmer, A. Sack, M. Heckel, P. Müller, T. Pöschel, Phys Rev E 2011, 84, 011301.
[47] M. N. Bannerman, L. Lue, J Chem Phys 2010, 133, 124506.
[48] M. N. Bannerman, L. Lue, J Chem Phys 2009, 130, 164507.
[49] W. G. Hoover, C. G. Hoover, M. N. Bannerman, J Stat Phys 2009, 136, 715.
[50] M. N. Bannerman, J. Magee, L. Lue, Phys Rev E 2009, 80, 021801.
[51] M. Marin, P. Cordero, In Proceedings of the 8th Joint EPS-APS International Conference on Physics Computing, In P. Borcherds, and M. Bubak, Eds., World Scientific, 1996, 315.
[52] A. T. Krantz, TOMACS 1996, 6, 185.
[53] A. Vrabecz, G. Tóth, Mol Phys 2006, 104, 1843.
[54] D. R. Jefferson, TOPLAS 1985, 7, 404.
[55] M. Marin, P. Cordero, Comput Phys Commur 1995, 92, 214.
[56] G. Paul, J Comput Phys 2007, 221, 615.
[57] M. P. Allen, D. J. Tildesley, Computer Simulation of Liquids; Oxford University Press: Bristol, 1987.
[58] R. Raman, D. S. Wise, IEEE Trans Comput 2008, 57, 567.
[59] M. D. Adams, D. S. Wise, ACM SIGPLAN Notices, 2006, 41, 39.
[60] T. Pöschel, T. Schwager, Computational Granular Dynamics; Springer: New York, 2005.
[61] M. D. Atkinson, J.-R. Sack, N. Santoro, T. Strothotte, Commun ACM 1986, 29, 996.
[62] H. C. Andersen, J Chem Phys 1980, 72, 2384.
[63] J. Weidendorfer, M. Kowarschik, C. Trinitis, In Computational Science—ICCS 2004; Springer-Verlag: Berlin, 2004, 3038, 440.
[64] A. Donev, S. Torquato, F. Stillinger, J Comput Phys 2005, 202, 737.
[65] D. Frenkel, J. F. Maguire, Mol Phys 1983, 49, 503.
[66] R. van Zon, J. Schofield, J Chem Phys 2008, 128, 154119.
[67] M. Marin, Comput Phys Commun 1997, 102, 81.
[68] S. Miller, S. Luding, J Comput Phys 2004, 193, 306.
[69] R. H. Swendsen, J. Wang, Phys Rev Lett 1986, 57, 2607.
[70] K. Erleben, Module based design for rigid body simulators Technical report, University of Copenhagen, 2002.
[71] L. V. Woodcock, Ann NY Acad Sci 1981, 371, 274.
[72] J. M. Haile, Molecular Dynamics Simulation—Elementary Methods; Wiley-Interscience: New York, 1997.
[73] M. A. Hopkins, H. H. Shen, J Fluid Mech 1992, 244, 477.
[74] A. W. Lees, S. F. Edwards, J Phys C 1972, 5, 1921.
[75] S. W. Smith, C. K. Hall, B. D. Freeman, J Comput Phys 1997, 134, 16.
[76] G. A. Bird, Molecular gas dynamics and the direct simulation of gas flows; Oxford University Press: Oxford, 1994.

Received: 20 April 2011
Revised: 4 July 2011
Accepted: 25 July 2011
Published online on 29 August 2011