# Multiple Time Step Integrators and Momentum Conservation

A. Kopf* and W. Paul

Institut für Physik, Johannes–Gutenberg–Universität,

D–55099 Mainz, Germany

B. Dünweg

Max–Planck–Institut für Polymerforschung,

Ackermannweg 10, D–55128 Mainz, Germany

**Abstract**

By use of the standard Liouville operator formalism, we derive a new symplectic multiple time step integrator for Hamiltonian systems with disparate masses, which, in contrast to previous algorithms, conserves the total momentum exactly, and is only moderately slower. The new scheme is tested numerically by application to Molecular Dynamics simulations of a polymer melt whose monomers have different masses, and compared to earlier algorithms.

**PACS:** 02.70.Ns, 61.25.Hq

**Keywords:** Molecular Dynamics, Multiple Time Step Methods
Liouville Operator Formalism, Symplectic Integrators
Verlet Algorithm, Momentum Conservation
Polymer Melts

*to whom correspondence should be addressed (phone: +49–6131–39–3645, fax: +49–6131–39–5441, e–mail: kopf@leonow.physik.uni-mainz.de).

# 1 Introduction

Classical Molecular Dynamics (MD) simulations of molecular systems [1–5] are frequently confronted with numerical difficulties related to the existence of a variety of physical time scales. For polymer systems, for instance, there is a hierarchy of frequencies: bond stretching, bond angle vibrations, librations in the minima of the torsion potential, and "Einstein frequencies" in the cage formed by the non–bonded interaction with the neighboring molecules. For a mixture of different molecules one may also have a large separation of time scales due to different masses. The numerical difficulty arises from the fact that the largest possible time step of a simple integrator like the standard Verlet algorithm [1] is governed by the highest frequency in the system, or the fastest degrees of freedom. This can be quite inefficient if the number of these degrees of freedom is small compared to the overall number of degrees of freedom. In order to cope with this problem, several authors have suggested to introduce a hierarchy of time steps just corresponding to the hierarchy of frequencies in the system [6–14]. The idea is to update the slow degrees of freedom less frequently than the fast ones, and thus save computer time.

A more recent development is the combination of this idea with the notion of symplectic and time–reversible integrators. These integrators [15] exhibit extraordinarily good stability, which is directly related to their mathematical properties: Except for roundoff errors, they exactly conserve the phase–space volume for each pair of coordinate and momentum separately (symplecticity), and bring the system back into the initial phase space configuration after turning the particles' velocities around (time–reversal symmetry). The stability can be simply explained from the fact that a global drift in, say, the total energy would mark the two directions of time as non–equivalent (which is possible as a result of roundoff errors, but not of discretization errors). While it had been known for quite a long time that the Verlet algorithm is actually the lowest–possible order symplectic time–reversible integrator, and more complicated higher–order algorithms had been developed [16, 17], symplecticity and time reversibility had not been combined with the multiple time step idea until recently, when Tuckerman *et al.* [12] suggested the so–called "r–RESPA" scheme. The most straightforward way to derive symplectic time–reversible integrators is based on the Liouville operator formalism, which has also been used in Ref. [12].

The usefulness of the "r–RESPA" algorithm has been demonstrated in many applications [18–22], and speed–ups by an order of magnitude are possible in favorable cases [20]. Hence, these new methods allow the study of complex systems which were previously inaccessible to MD. Moreover, the method has been extended to Car–Parrinello type simulations [23–25], and combined with both the Nose–Hoover thermostat [26] and the Parrinello–Rahman constant pressure algorithm [27].

In this paper, we are concerned with the simple special case of different time scales being introduced just by disparate masses. This case has been treated in Ref. [12], but the scheme considered there does not conserve the total momentum of the system. Since however total momentum conservation is directly related to the correct simulation of the long–wavelength, long–time hydrodynamic properties of the system [28], we consider an algorithm which is capable of momentum conservation rather important. It is the purpose of the present paper to show that a suitably modified multiple time step algorithm does conserve the total momentum exactly in each integration step, and is not much more time–consuming than the original scheme.

The paper is organized as follows: In Sec. 2, we use the Liouville operator formalism to derive the integrators, which in Sec. 3 are examined with respect to their stability and their performance, while Sec. 4 summarizes our conclusions.

# 2 Reversible Integrators

## 2.1 Liouville Operator Formalism

Denoting the position of the $j$th particle by $\vec{r}_j$, its mass and momentum by $m_j$ and $\vec{p}_j$, respectively, and the total force acting on it by $\vec{F}_j$, the Liouville operator is written as

$$i\hat{L} = \sum_j \left[ \frac{\vec{p}_j}{m_j} \frac{\partial}{\partial \vec{r}_j} + \vec{F}_j \frac{\partial}{\partial \vec{p}_j} \right] . \tag{1}$$

The time development of an arbitrary observable $A\left(\{\vec{r}_i\}, \{\vec{p}_i\}\right)$ is then given by the differential equation

$$\frac{d}{dt} A = i\hat{L} A \tag{2}$$

or its formal solution

$$A(t) = \exp\left(i\hat{L}t\right) A(0), \tag{3}$$

where for simplicity we use the notation $A(t)$ for $A\left(\{\vec{r}_i(t)\}, \{\vec{p}_i(t)\}\right)$. The conservation of the phase–space volume follows directly from the Hermiticity of $\hat{L}$,

$$\int d\vec{r}d\vec{p} \quad f^\star(\hat{L}g) \quad = \quad \int d\vec{r}d\vec{p} \quad (\hat{L}f)^\star g, \tag{4}$$

which implies that the time–evolution operator $\exp\left(i\hat{L}t\right)$ is unitary. Similarly, the time–inversion symmetry is directly read off from

$$\exp\left(i\hat{L}t\right) \exp\left(-i\hat{L}t\right) = 1. \tag{5}$$

## 2.2 Verlet Algorithm

Since $\hat{L}$ is the sum of non–commuting operators, a numerical implementation requires that $\exp\left(i\hat{L}t\right)$ is factorized in an appropriate way. For a small time step $\Delta t$, the following approximation is correct up to quadratic order in $\Delta t$:

$$\begin{aligned} \exp\left(i\hat{L}\Delta t\right) &\rightarrow V\left(\exp\left(i\hat{L}\Delta t\right)\right) \\ &= \exp\left(i\hat{L}_p\Delta t/2\right) \exp\left(i\hat{L}_r\Delta t\right) \exp\left(i\hat{L}_p\Delta t/2\right), \end{aligned} \tag{6}$$

where the abbreviations

$$i\hat{L}_p = \sum_j \vec{F}_j \frac{\partial}{\partial \vec{p}_j} \tag{7}$$

$$i\hat{L}_r = \sum_j \frac{\vec{p}_j}{m_j} \frac{\partial}{\partial \vec{r}_j} \tag{8}$$

(i. e. the decomposition of $\hat{L}$ in position part and momentum part) have been introduced, and $V$ stands for the Verlet–type factorization of the time development operator. This factorization manifestly induces a time–reversal symmetric and symplectic algorithm, which is the velocity Verlet updating scheme:

$$\vec{r}_j(t+\Delta t) = \vec{r}_j(t) + \Delta t \frac{\vec{p}_j(t)}{m_j} + \frac{(\Delta t)^2}{2m_j} \vec{F}_j\left(\{\vec{r}_i(t)\}\right) \tag{9}$$

$$\vec{p}_j(t+\Delta t) = \vec{p}_j(t) + \frac{\Delta t}{2}\left[\vec{F}_j\left(\{\vec{r}_i(t)\}\right) + \vec{F}_j\left(\{\vec{r}_i(t+\Delta t)\}\right)\right]. \tag{10}$$

Since in the production phase the algorithm is nothing but an alternating application of $\exp\left(i\hat{L}_r\Delta t\right)$ and $\exp\left(i\hat{L}_p\Delta t\right)$, it is equivalent to the other possible construction, where $\hat{L}_r$ and $\hat{L}_p$ are exchanged.

In order to discuss momentum conservation in the Verlet algorithm, we note that from the above equations one finds for the updating of the total momentum

$$\begin{aligned} & V\left(\exp\left(i\hat{L}\Delta t\right)\right)\sum_j \vec{p}_j(t) = \sum_j \vec{p}_j(t+\Delta t) \\ = & \sum_j \vec{p}_j(t) + \frac{\Delta t}{2}\sum_j \left[\vec{F}_j\left(\{\vec{r}_i(t)\}\right) + \vec{F}_j\left(\{\vec{r}_i(t+\Delta t)\}\right)\right]. \end{aligned} \tag{11}$$

Hence, the total momentum is exactly conserved if the forces always add up to zero, as they do if the system is not coupled to an external field (Newton's third law).

## 2.3 Multiple Time Step Integrators

For the construction of multiple time step algorithms we consider for simplicity only two types of degrees of freedom, the "slow" ones (s), and the "fast" ones (f). Assuming that $\hat{L}$ can be decomposed as

$$\hat{L} = \hat{L}_f + \hat{L}_s, \tag{12}$$

where $\hat{L}_f$ is associated with the high frequencies in the system (the "fast" degrees of freedom), and $\hat{L}_s$ with the low ones, two possible symplectic and time–reversible multiple time step integrators based on the Verlet algorithm are

- FastSlowFast:

$$\begin{aligned} & \exp\left(i\hat{L}\Delta t\right) \rightarrow \\ & \left[V\left(\exp\left(i\hat{L}_f\delta t\right)\right)\right]^{n/2} V\left(\exp\left(i\hat{L}_s\Delta t\right)\right)\left[V\left(\exp\left(i\hat{L}_f\delta t\right)\right)\right]^{n/2}, \end{aligned} \tag{13}$$

- SlowFastSlow:

$$\begin{aligned} & \exp\left(i\hat{L}\Delta t\right) \rightarrow \\ & V\left(\exp\left(i\hat{L}_s\Delta t/2\right)\right)\left[V\left(\exp\left(i\hat{L}_f\delta t\right)\right)\right]^{n} V\left(\exp\left(i\hat{L}_s\Delta t/2\right)\right). \end{aligned} \tag{14}$$

Here we have assumed that the "fast" frequencies are roughly a factor of $n$ higher than the "slow" ones, such that numerical stability requires to update the "fast" degrees of freedom with

a time step which is $n$ times smaller than that of the slow variables, $\delta t = \Delta t/n$. In terms of explicit programming, the FastSlowFast algorithm involves

- $n/2$ time steps $\delta t$ with the propagator for the fast degrees of freedom
- one time step $\Delta t$ with the propagator for the slow degrees of freedom
- $n/2$ time steps $\delta t$ with the propagator for the fast degrees of freedom.

Conversely, the SlowFastSlow scheme means explicitly

- one time step $\Delta t/2$ with the propagator for the slow degrees of freedom
- $n$ time steps $\delta t$ with the propagator for the fast degrees of freedom
- one time step $\Delta t/2$ with the propagator for the slow degrees of freedom.

Note that the details of the decomposition of $\hat{L}$ have been left open so far, and that the two schemes are really different from each other: While in both schemes the updating of the "fast" degrees of freedom is just a sequence of $n$ Verlet integrations with $\hat{L}_f$ and time step $\delta t$, the slow degrees of freedom are treated differently in the two schemes. SlowFastSlow involves two consecutive integrations with $\hat{L}_s$ and time step $\Delta t/2$, while FastSlowFast is just a single integration with time step $\Delta t$. It is obvious that the latter scheme is hence less accurate and also less stable. The price to be paid for the additional accuracy of SlowFastSlow is of course one more CPU time expensive force calculation (assuming a majority of slow degrees of freedom). For the detailed discussion of this point we refer to the next section.

We now specialize to a system with two masses, i. e. $m_j = m_f$ for $j = 1, \ldots, M$, and $m_j = m_s > m_f$ for $j = M+1, \ldots, N$. The decomposition of $\hat{L}$ suggested by Tuckerman *et al.* [12] is

$$i\hat{L}_f = \sum_{j=1}^{M} \left[ \frac{\vec{p}_j}{m_f} \frac{\partial}{\partial \vec{r}_j} + \vec{F}_j \frac{\partial}{\partial \vec{p}_j} \right] \tag{15}$$

$$i\hat{L}_s = \sum_{j=M+1}^{N} \left[ \frac{\vec{p}_j}{m_s} \frac{\partial}{\partial \vec{r}_j} + \vec{F}_j \frac{\partial}{\partial \vec{p}_j} \right] . \tag{16}$$

It is quite obvious that this scheme does not conserve the total momentum of the system. As discussed in the previous subsection, the total momentum is changed by an amount

$(\delta t/2)\sum_{j=1}^{M}\left[\vec{F}_j(t)+\vec{F}_j(t+\delta t)\right]$ in a "fast" updating step, and this does not vanish, since the slow particles produce an effective external field which acts upon the fast particles. Since after these updates the fast particles have changed their position, the forces which they later on (during a "slow" update) exert on the slow particles do not cancel these contributions. Since the total momentum displacements produced by this effect in a liquid should, for sufficiently long times, behave randomly, one should expect diffusive behavior of the total momentum. This is indeed observed, as demonstrated in the next section.

Conversely, if one succeeds in finding a decomposition of the Liouville operator such that both $i\hat{L}_f\sum_{j=1}^{N}\vec{p}_j=0$ and $i\hat{L}_s\sum_{j=1}^{N}\vec{p}_j=0$, then it is immediately obvious that such a decomposition would yield an algorithm which does conserve the total momentum. A decomposition which satisfies this condition, and which we propose in the present paper, is

$$i\hat{L}_f = \sum_{j=1}^{N}\frac{\vec{p}_j}{m_j}\frac{\partial}{\partial\vec{r}_j}+\sum_{j=1}^{M}\vec{F}_j\frac{\partial}{\partial\vec{p}_j}+\sum_{j=M+1}^{N}\vec{F}_j^{fast}\frac{\partial}{\partial\vec{p}_j} \tag{17}$$

$$i\hat{L}_s = \sum_{j=M+1}^{N}\vec{F}_j^{slow}\frac{\partial}{\partial\vec{p}_j}, \tag{18}$$

where the forces on the slow particles ($j=M+1,\ldots,N$) have been split up into a first part $\vec{F}_j^{fast}$, which is induced by the fast particles, and a second part $\vec{F}_j^{slow}$, induced by other slow particles. The idea behind this decomposition is the observation that a multiple time step integrator will only yield a substantial speed–up if the slow particles are a large fraction of the overall system. Under these circumstances, however, most of the CPU time goes into calculating the forces between these slow particles. If it is possible to do this operation only every $n$th step, i. e. if $\hat{L}_s$ contains all of the forces between the slow particles, most of the possible gain is achieved. That both $\hat{L}_s$ and $\hat{L}_f$ indeed conserve the total momentum separately is very easily seen from the fact that $i\hat{L}_s\sum_j\vec{p}_j=0$ as a result of Newton III within the subsystem of slow particles, while $i\hat{L}_f\sum_j\vec{p}_j=0$ must hold since also $i\hat{L}\sum_j\vec{p}_j=0$ and $\hat{L}_f=\hat{L}-\hat{L}_s$. It should be noted that for this special decomposition $V\left(\exp\left(i\hat{L}_s\Delta t\right)\right)=\exp\left(i\hat{L}_s\Delta t\right)$, such that FastSlowFast and SlowFastSlow algorithms coincide (this is not so for the decomposition by Tuckerman *et al.*). The proposed algorithm then amounts to an alternation between

- $n$ Verlet integration steps with time step $\delta t$, where all particles are propagated and all forces are taken into account, except those which act between the slow particles, and

- one propagation of the velocities of the slow particles, where the big time step $\Delta t$ is used, and only the forces between these slow particles are taken into account.

# 3 Numerical Tests

## 3.1 Model System and Simulation Details

The model system for our simulation is a melt of bead–spring polymer chains [29]. One chain consists of $N$ monomers, held together by the FENE potential

$$U^{ch}(r) = -0.5kR_0^2 \ln\left[1 - (r/R_0)^2\right]. \tag{19}$$

Furthermore, a shifted purely repulsive Lennard–Jones potential

$$U^{LJ}(r) = 4\epsilon\left[(\sigma/r)^{12} - (\sigma/r)^6 + \frac{1}{4}\right], \tag{20}$$

truncated at $r = 2^{1/6}\sigma$, acts between all monomers in order to model the excluded–volume interaction. The parameters $R_0 = 1.5\sigma$ and $k = 30\epsilon/\sigma^2$ were chosen to ensure that bond–crossing is practically impossible [29]. We use Lennard–Jones units where $\epsilon = \sigma = 1$, and the mass of the *fast* monomers is also set equal to unity. Within a chain, all monomers have the same mass. Runs were performed at monomer number density $\rho = 0.85$, temperature $k_BT = 1.0$, and chain length $N = 20$. In order to have a large fraction of slow degrees of freedom, 24 chains were assigned monomer mass $m_s$, while the remaining 6 chains consisted of light monomers ($m = m_f = 1$). Apart from the different masses, the chains are identical. We studied the cases $m_s = 16$ as well as $m_s = 100$, and always used a time step for the fast integration of $\delta t = 0.003$, which ensures stability for runs with several million time steps. For our numerical tests, we however confined ourselves to $2 \times 10^4$ steps per run, where we define one step as one integration with $\delta t$. The results were then averaged over 10 independent runs. Since the typical oscillation time is proportional to the square root of the involved mass, we used at most $n_{max} = \sqrt{m_s/m_f}$ fast integrations per slow integration, but also studied cases $n < n_{max}$.

## 3.2 Violation of Momentum Conservation for the Tuckerman *et al.* Decomposition

Figure 1 shows the x– and y–component of the total momentum of the system using Tuckerman's algorithm in its FastSlowFast variant for a mass ratio of $m_s/m_f = 100$, and $n = 10$. Only the first 1000 steps are shown. Qualitatively, this looks much like the expected random–walk–like motion. More quantitatively, we also calculated the mean squared displacement of the total momentum, which is plotted vs. time (in MD steps) in Fig. 2. Indeed, for long times the displacement is proportional to time. i. e. behaves diffusively. For short times (see inset of Figure 2) we observe some memory effects of the walk, resulting in a peak at short times. Apparently, the second and third cycle tend to "correct" the error introduced by the first one, but not fully, such that a net diffusion remains. It is evident from our data (the corresponding plot is not shown here) and in agreement with physical intuition that the violation of momentum conservation is the larger the higher the number of small steps $n$ per big step is.

## 3.3 Stability

Before looking in detail at energy conservation for different multiple time step integration schemes, we first discuss these properties in a qualitative way. It is useful to look at the normalized deviation from the initial energy, i. e.

$$\frac{\Delta E(t)}{E(0)} = \frac{E(t) - E(0)}{E(0)}. \tag{21}$$

Figure 3 compares this energy deviation for the two propagators proposed by Tuckerman *et al.* with the one discussed in this work. All of these runs were actually done with exactly the same initial configuration leading to the same structure in the energy fluctuations. This indicates that the integrators produce at the beginning very similar trajectories in phase space. The small deviations stemming from different integrators will grow exponentially with time, and at some point all correlations between the trajectories will vanish [15]. We now proceed with a more quantitative discussion of energy conservation.

The numerical fluctuations of the different integrators can be characterized by the following two quantities [26]

$$\Delta E^{rms} = \frac{\Delta E_{total}}{\Delta E_{kin}}, \tag{22}$$

$$\Delta E \quad = \quad \frac{1}{T}\sum_{t=1}^{T}\left|\frac{E(t)-E(0)}{E(0)}\right|. \tag{23}$$

Here $\Delta E_{total}$ and $\Delta E_{kin}$ are the root mean square fluctuations of the total energy and the kinetic energy, respectively. So $\Delta E^{rms}$ is the ratio of the numerical fluctuations of the total energy due to the finite order of the integrator (which should of course be as small as possible) to the fluctuations of the kinetic energy, which are inherent in a microcanonical simulation and related to the specific heat. $\Delta E$ is simply the time average of the absolute value of the deviation $\Delta E(t)/E(0)$ as defined above. The data which we obtained are listed in Tables 1 and 2.

First it is noticeable that the order of magnitude of energy fluctuations is the same for all integration schemes. As one expects, the Verlet algorithm leads to the smallest fluctuations. Common to all multiple time step algorithms discussed here is a decreasing stability with higher $n$. With respect to stability our algorithm is worse than the SlowFastSlow variant but better than the FastSlowFast variant of the Tuckerman *et al.* decomposition. The reason for SlowFastSlow showing better stability than the others has already been discussed in the previous section.

## 3.4 Performance

In order to obtain a good improvement in CPU time by multiple time step methods, it is necessary to have a clear separation of time scales. The other factor which determines the speed–up is the ratio of the number of fast to slow degrees of freedom. Since the multiple time step approach rests on saving the operations related to the slow degrees of freedom (in particular, the force calculation within the subsystem of slow particles), it is useless to employ it for systems with only a few slow degrees of freedom. Considerable net gains can only be expected if the fraction of light particles is of the order of 10% or less. This is demonstrated in Table 3, where we list the CPU time needed for $10^5$ steps for the various algorithms, confining ourselves to the case $m_s/m_f = 100$, $n = 10$, but varying the fraction of light chains. The runs were done on one processor of a Convex/HP SPP1200 machine. The data clearly show that the speed–up is only moderate even in the case of only a single light chain.

Of course, it is not possible to give a general quantitative statement about the performance gains of multiple time step methods, since these depend in a non–trivial way on the model (i. e. the range of interaction, the form of the potential, the system size, the density, etc.), the

details of the computer architecture (e. g. scalar, vector or parallel machine, cache size, etc.), the quality of the compiler, and the details of the implementation. We used a scalar version of a link–cell algorithm combined with a Verlet table [30], and optimized the "skin" parameter (cf. Ref. [30]) for each algorithm separately. A program using different techniques will definitely yield other speed–up factors. Moreover, we expect much better improvements if the interaction is more complicated (i. e. computationally more expensive) than our simple potentials. The fact that the CPU time increases with the fraction of light particles is easily understandable: The simple Verlet algorithm needs more frequent updates of the neighbor table for a lighter, i. e. faster system, and the multiple time step methods spend more time in the integration steps for the fast degrees of freedom.

Quite generally, it is clear that SlowFastSlow is slower than FastSlowFast, due to the additional force calculation for the slow degrees of freedom which are the majority. Moreover, the new algorithm discussed in the present paper is usually also somewhat slower than FastSlowFast, since one has to integrate the positions and velocities of *all* particles in the inner ("fast") loop, in contrast to updating only the fast degrees of freedom. Hence the loss compared to FastSlowFast is determined by the CPU time needed to do these updates, which is usually moderate in comparison with the force calculation. The measurements confirm that the algorithm proposed in this paper is somewhere between the two others, and the difference in performance between all three variants is not very big.

# 4 Conclusions

We have shown in this work how to construct a reversible multiple time step integrator that exactly conserves the total momentum in each integration step. This was done by taking into account Newton's third law in the decomposition of the Liouville operator into a fast and a slow part, such that each part satisfies the conservation law separately. The resulting algorithm has stability and performance properties comparable to those of multiple time step algorithms described in the literature. For all algorithms, the speed–up factor depends crucially on the fraction of CPU time which is spent in the force routine for the slow degrees of freedom. We found that under reasonable conditions it can be much smaller than the optimistic values reported in the literature [10]; nevertheless we feel that with the development of these algorithms

a significant progress in MD methodology has been achieved.

# 5 Acknowledgement

A. K. acknowledges support by the German federal department for education and research (BMBF) under Grant No. 03–BI4MAI–7.

# References

[1] Allen, M. P. and Tildesley, D. J., *Computer Simulation of Liquids*, Clarendon, Oxford, 1987.

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

[3] Ciccotti, G. and Hoover, W. G., editors, *Molecular–Dynamics Simulation of Statistical–Mechanical Systems*, North–Holland, Amsterdam, 1986.

[4] Allen, M. P. and Tildesley, D. J., editors, *Computer Simulation in Chemical Physics*, Kluwer Academic Publishers, Dordrecht, 1993.

[5] Binder, K. and Ciccotti, G., editors, *Monte Carlo and Molecular Dynamics of Condensed Matter Systems*, Italian Physical Society, Bologna, 1996.

[6] Streett, W. B., Tildesley, D. J., and Saville, G., Mol. Phys. **35** (1978) 639.

[7] Swindoll, R. D. and Haile, J. M., J. Comp. Phys. **53** (1984) 289.

[8] Teleman, O. and Jönsson, B., J. Comp. Chem. **7** (1986) 58.

[9] Tuckermann, M. E., Martyna, G. J., and Berne, B. J., J. Chem. Phys. **93** (1990) 1287.

[10] Tuckermann, M. E., Berne, B. J., and Rossi, A., J. Chem. Phys. **94** (1991) 1465.

[11] Tuckerman, M. E. and Berne, B. J., J. Chem. Phys. **95** (1991) 4389.

[12] Tuckerman, M. E., Berne, B. J., and Martyna, G. J., J. Chem. Phys. **97** (1992) 1990.

[13] Scully, J. L. and Hermans, J., Molecular Simulation **11** (1993) 67.

[14] Forester, T. and Smith, W., Molecular Simulation **13** (1994) 195.

[15] M. Sprik, in Ref. [5].

[16] Yoshida, H., Physics Letters A **150** (1990) 262.

[17] Gray, S. K., Noid, D. W., and Sumpter, B. G., J. Chem. Phys. **101** (1994) 4062.

[18] Tuckerman, M. E. and Berne, B. J., J. Chem. Phys. **98** (1993) 7301.

[19] Tuckerman, M. E. and Langel, W., J. Chem. Phys. **100** (1994) 6368.

[20] Procacci, P. and Berne, B. J., J. Chem. Phys. **101** (1994) 2421.

[21] Mizan, T. I., Savage, P. E., and Ziff, R. M., J. Phys. Chem. **98** (1994) 13067.

[22] Humphrey, D. D., Friesner, R. A., and Berne, B. J., J. Phys. Chem. **98** (1994) 6885.

[23] Tuckerman, M. E. and Parrinello, M., J. Chem. Phys. **101** (1994) 1302.

[24] Tuckerman, M. E. and Parrinello, M., J. Chem. Phys. **101** (1994) 1316.

[25] Gibson, D. A. and Carter, E. A., J. Phys. Chem. **97** (1993) 13429.

[26] Watanabe, M. and Karplus, M., J. Chem. Phys. **99** (1993) 8063.

[27] Procacci, P. and Berne, B. J., Mol. Phys. **83** (1994) 255.

[28] Dünweg, B., J. Chem. Phys. **99** (1993) 6977.

[29] Kremer, K. and Grest, G. S., J. Chem. Phys. **92** (1990) 5057.

[30] Grest, G. S., Dünweg, B., and Kremer, K., Comp. Phys. Comm. **55** (1989) 269.

## Figures

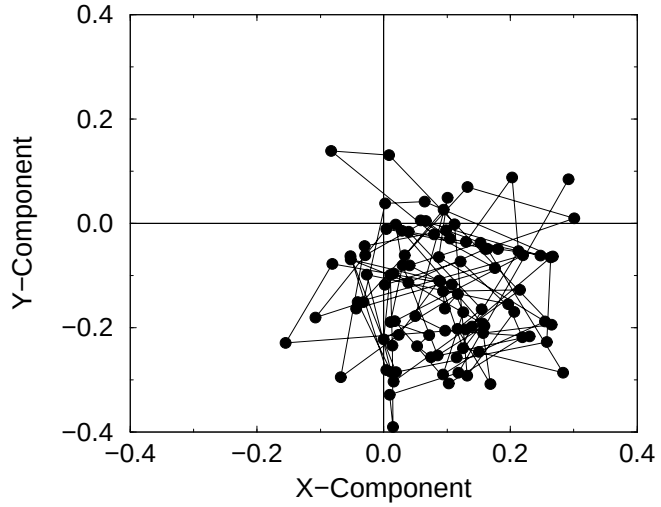


Figure 1: x– and y–components of the total momentum in LJ units for the first 1000 MD steps with Tuckerman's FastSlowFast algorithm, for mass ratio $m_s/m_f = 100$, and $n = 10$.

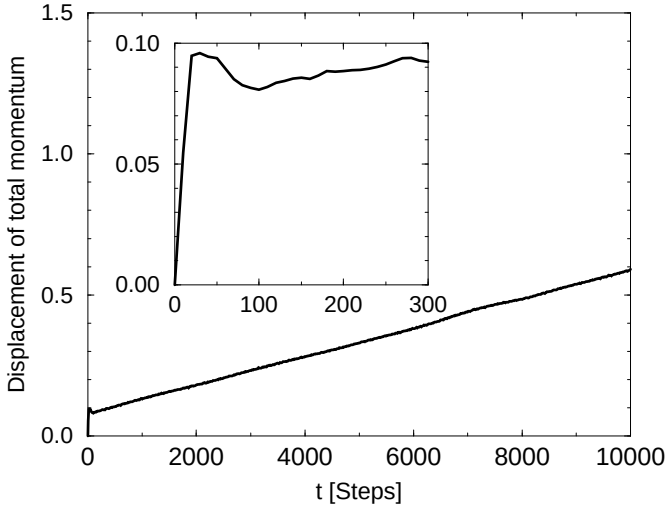


Figure 2: Mean squared displacement of the total momentum as a function of time (measured in number of small integration steps) with Tuckerman's FastSlowFast algorithm for $n = 10$ small steps per big integration step.

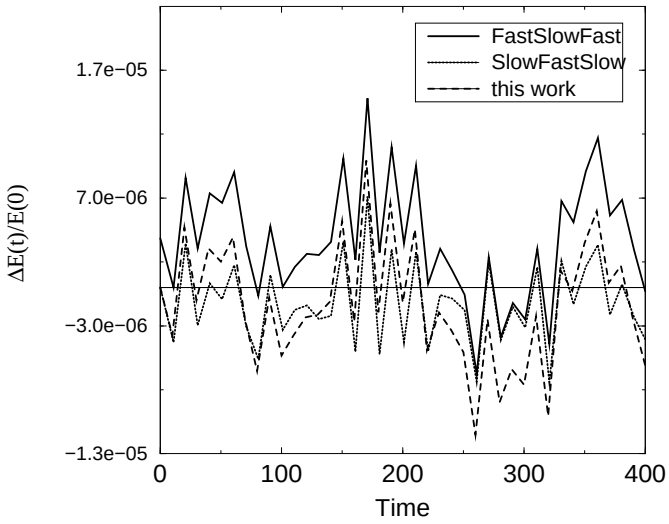


Figure 3: $\Delta E(t)/E(0)$ as a function of time (in MD steps) for different multiple time step algorithms starting from the same initial configuration.

## Tables

Table 1: $\Delta E^{rms}$ and $\Delta E$, characterizing the integrators' stability, for a mass ratio of $m_s/m_f = 16$ and various numbers of small steps per big step, $n$. For this mass ratio, $n = 4$ corresponds to the natural time scaling.

| $m = 16$ | Verlet | this work | | Tuckerman FastSlowFast | | Tuckerman SlowFastSlow | |
|---|---|---|---|---|---|---|---|
| | | $n = 2$ | $n = 4$ | $n = 2$ | $n = 4$ | $n = 2$ | $n = 4$ |
| $\Delta E^{rms} * 100$ | 0.1868 | 0.1995 | 0.3900 | 0.2152 | 0.5000 | 0.1879 | 0.2162 |
| $\Delta E * 10^5$ | 0.2736 | 0.3100 | 0.6032 | 0.3471 | 0.8609 | 0.3145 | 0.3473 |

Table 2: $\Delta E^{rms}$ and $\Delta E$ for a system with a mass ratio of $m_s/m_f = 100$, i. e. a maximum number of small time steps of $n = 10$.

| $m = 100$ | Verlet | this work | | | Tuckerman FastSlowFast | | | Tuckerman SlowFastSlow | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | $n = 4$ | $n = 8$ | $n = 10$ | $n = 4$ | $n = 8$ | $n = 10$ | $n = 4$ | $n = 8$ | $n = 10$ |
| $\Delta E^{rms} \times 100$ | 0.1849 | 0.1938 | 0.2896 | 0.3971 | 0.1928 | 0.3041 | 0.4423 | 0.1834 | 0.1971 | 0.2072 |
| $\Delta E \times 10^5$ | 0.3647 | 0.4380 | 0.5470 | 0.6723 | 0.4206 | 0.5768 | 0.8350 | 0.4134 | 0.4370 | 0.4606 |

Table 3: CPU time in seconds needed by all discussed algorithms for $10^5$ steps for different mixing ratios $r$, i. e. the ratio of the number of light chains to the total number of chains. Times for multiple time step algorithms are relative to the Verlet algorithm.

| $m = 100$, $n = 10$ | Verlet | this work | FastSlowFast | SlowFastSlow |
|---|---|---|---|---|
| $r = 3.3\%$ | 400 | 0.60 | 0.57 | 0.64 |
| $r = 10.0\%$ | 413 | 0.67 | 0.64 | 0.76 |
| $r = 20.0\%$ | 432 | 0.72 | 0.73 | 0.84 |