Efficient constraint dynamics using MILC SHAKE

A.G. Bailey [a], C.P. Lowe [b], A.P. Sutton [a]

[a] *Department of Physics, Imperial College London, South Kensington Campus, Prince Consort Road, London SW7 2AZ, United Kingdom*

[b] *HIMS, Universiteit van Amsterdam, Nieuwe Achtergracht 188, 1018 WV Amsterdam, The Netherlands*

Abstract

We address the problem of imposing rigid constraints between connected sites in a dynamic computer simulation. For two important cases, the linear and ring topologies, each site is connected to at most two nearest neighbors. The constraint matrix is then invertible in order n operations. We show that, this being the case, a computational method based on a matrix inversion of the linearized constraint equations (MILC SHAKE) can be orders of magnitude faster than the simple SHAKE or RATTLE methods.

Key words: Constraints, Lagrange multipliers, SHAKE, RATTLE, Molecular dynamics, Simulations
PACS: 70H45, 70-08, 70E60, 90C53, 65H10

1. Introduction

Imposing internal geometric constraints in dynamic simulations is a generic computational problem. The most well known example is in molecular dynamics, where chemical bonds are modeled by fixing the distance between atoms. This integrates out the fast vibrational degrees of freedom associated with the stiff bonding potential. Consequently, much longer time-steps are possible and a faster sampling of phase space. On a totally different length scale, the identical problem arises in the fields of robotics [1], computer animation [2] and virtual reality [3]. The formalism for tackling the problem numerically

Email addresses: aimee.bailey06@imperial.ac.uk (A.G. Bailey), lowe@science.uva.nl (C.P. Lowe), a.sutton@imperial.ac.uk (A.P. Sutton).

was derived by Ryckaert *et al.* [4]. For a system with n constraints, a set of n non-linear equations must be solved. These equations can be linearized and solved iteratively. On the grounds that matrix inversion is an order n^3 process, they developed a simple alternative scheme termed SHAKE. In the SHAKE algorithm successive linearized constraints are satisfied in order n operations. This procedure violates constraints already satisfied but the error generally shrinks and, applied iteratively, the scheme usually converges. In a dynamic scheme one is often interested in velocities as well as positions. The condition that the relative velocity along a constraint is zero can be treated in a similar manner. One notable difference is that this involves solving n equations that are linear. If these linear equations are solved iteratively, in the same way as with SHAKE, the combination of the two algorithms is termed RATTLE [5].

While this methodology is quite general, the iterative scheme can be quite slowly convergent. For problems involving small numbers of constraints, in MD terms small molecules, alternative methods are more efficient. For example, for water molecules the SETTLE algorithm [6] essentially uses an analytic solution for the constraints problem. Kräutler *et al.* also showed that solving the full order n^3 problem can be more efficient than SHAKE for small molecules with their M-SHAKE algorithm [7]. Gonnet developed the P-SHAKE method [8] that uses a pre-conditioner to reduce the computational effort per iteration to order n^2 operations. Barth *et al.* found that by using successive over-relaxation techniques the calculation overhead for SHAKE could be reduced by a factor of two [9]. They further investigated algorithms using Newton-like iteration combined with sparse matrix inversion and found that fewer iterations were required and in some cases there was a moderate improvement in overall performance. However, Lowe and de Leeuw pointed out that under certain circumstances the constraint matrix is not only sparse, it is tridiagonal [10]. A tridiagonal matrix is trivially inverted in order n operations [11]. The condition required is that we have a "linear chain". That is, a chain in which only successive sites are connected. The linear chain is encountered in several situations. An example in the field of MD is the simulation of alkanes using a united atom model [12]. Similar methodology is also used for mesoscopic simulation of inextensible filaments of the type frequently encountered in biological systems [13]. For this class of problem one can ask the question, does direct matrix inversion compare with SHAKE and RATTLE? Promisingly, because the velocity constraint equations are linear, direct matrix inversion solves for the velocity constraints directly. No iteration is required. Iteration is only necessary to satisfy the positional constraints, whereas for RATTLE it is necessary for both.

Here we address this question by developing and testing a method for tackling this problem based on matrix inversion. It uses "Matrix Inverted Linearized Constraints" so we subsequently refer to it as MILC SHAKE. Two simple but non-trivial test cases are considered. One is satisfying the constraints in a dynamic simulation of an inextensible chain with resistance to bending. The other is satisfying the constraints following a random violation. For the former the violation of the constraints is not random but correlated. The results show that either way MILC SHAKE is a significant factor more efficient than SHAKE or RATTLE for small systems (ten total sites) and orders of magnitude better for larger systems (hundreds to thousands of sites).

Addressing the linear chain problem seems a little restrictive, but with a minor modification the same methodology applies for ring topologies. The reason is that the constraint matrix for a ring is straightforwardly diagonalized in order n operations. Therefore, the

problem is totally equivalent to the linear MILC SHAKE example. This means that the methodology can also be used for an important case where SHAKE and RATTLE are at best very slowly convergent and at worst non-convergent [14]. Furthermore, we speculate that for more general topologies MILC SHAKE could still be useful as part of a hybrid scheme.

2. Description of the algorithm

Let us start with a system of N sites, where the location of the i^{th} site at time t is signified by $\mathbf{r}_i(t)$. The aim is to rigidly constrain pairs of sites to be a specified distance apart. The vector between two sites that are rigidly connected will be referred to here as a *link*. The equations representing these distance constraints can be written in the form of n equations, where n is the number of constraints in the system. Defining $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ as the link vector and l_{ij} as the link length, they are

$$\sigma_{ij}(\{\mathbf{r}(t)\}) = \mathbf{r}_{ij}^2(t) - l_{ij}^2. \tag{1}$$

Using Lagrange's method of undetermined multipliers [15], we can enforce the constraint equations while integrating Newton's laws. After the introduction of a zero potential term, the equation of motion in Cartesian coordinates is

$$m_i \frac{d^2\mathbf{r}_i(t)}{dt^2} = -\frac{\partial}{\partial \mathbf{r}_i}\left[U(\{\mathbf{r}(t)\}) + \sum_{i\to p} \lambda_{ip}\sigma_{ip}(\{\mathbf{r}(t)\})\right], \tag{2}$$

where the summation is over all sites, indexed by p, connected to site i. $U(\{\mathbf{r}(t)\})$ is the potential energy of the system, and the variable m_i is the mass concentrated at the i^{th} site. The λ_{ip} terms represent the undetermined Lagrange multipliers between the two indexed sites. For clarity, the position-dependence of the quantity in brackets on the right hand side has not been written explicitly.

We can write the position after a time step Δ as the sum of the unconstrained positions, denoted by $\tilde{\mathbf{r}}$, plus the adjustment due to the constraint forces. To order Δ^2,

$$\mathbf{r}_i(t+\Delta) = \tilde{\mathbf{r}}_i(t+\Delta) + \frac{\Delta^2}{2m_i}\mathbf{F}_i^C(t). \tag{3}$$

These forces act along the link vectors. Following from Eq. (2), a constraint force will take the following form.

$$\mathbf{F}_i^C(t) = \frac{1}{\Delta^2}\sum_{i\to p}\lambda_{ip}\frac{\partial\sigma_{ip}(\{\mathbf{r}(t)\})}{\partial\mathbf{r}_i} = \frac{2}{\Delta^2}\sum_{i\to p}\lambda_{ip}\mathbf{r}_{ip}(t) \tag{4}$$

The factors of Δ are included for convenience. Now the updated positions are explicitly

$$\mathbf{r}_i(t+\Delta) = \tilde{\mathbf{r}}_i(t+\Delta) + \frac{1}{m_i}\sum_{i\to p}\lambda_{ip}\mathbf{r}_{ip}(t). \tag{5}$$

and the constraint equation for the link between i and j is

$$\mathbf{r}_{ij}^2(t+\Delta) = l_{ij}^2. \tag{6}$$

Substituting Eq. (5) into Eq. (6), we arrive at the system of equations that we need to solve:

$$\mathbf{r}_{ij}^2(t+\Delta) = l_{ij}^2 = \left[\tilde{\mathbf{r}}_{ij}(t+\Delta) + \frac{1}{m_i}\sum_{i\to p}\lambda_{ip}\mathbf{r}_{ip}(t) - \frac{1}{m_j}\sum_{j\to q}\lambda_{jq}\mathbf{r}_{jq}(t)\right]^2. \qquad (7)$$

The indices p and q cycle through all sites linked to sites i and j, respectively.

2.1. *SHAKE*

The best known method for solving this set of equations is the algorithm SHAKE [4], and the velocity Verlet version RATTLE [5]. Two approximations are made to calculate the solutions. First, the equations are decoupled. Each link is considered isolated and each λ_{ij} is solved as if the sites i and j are involved in no other constraint equations. Second, the equations are linearized. All terms of order λ^2 are set to zero. With these approximations, the set of equations is no longer exact, and an iterative procedure is required to solve for the Lagrange multipliers. We focus on the first link and calculate the SHAKE approximation to λ_{ij}.

$$\lambda_{ij} = \frac{\tilde{\mathbf{r}}_{ij}^2 - l_{ij}^2}{2\left(m_i^{-1} + m_j^{-1}\right)\mathbf{r}_{ij}\cdot\tilde{\mathbf{r}}_{ij}} \qquad (8)$$

Both sites i and j are updated according to the following equations.

$$\begin{aligned}\mathbf{r}_i(t+\Delta) &= \tilde{\mathbf{r}}_i(t+\Delta) + \frac{1}{m_i}\lambda_{ij}\mathbf{r}_{ij}(t)\\ \mathbf{r}_j(t+\Delta) &= \tilde{\mathbf{r}}_j(t+\Delta) - \frac{1}{m_j}\lambda_{ij}\mathbf{r}_{ij}(t)\end{aligned} \qquad (9)$$

The positions in Eq. (9) are now taken to be the new unconstrained site locations – the $\tilde{\mathbf{r}}_i$ terms. The next link is then adjusted in a similar fashion, regardless of whether the preceding constraint has been violated in the process. All of the constraints of the system are cycled through in this manner until every one has been once satisfied to linear order. The error of every link is then calculated, and if all constraints are satisfied to within a predefined error, the procedure is complete. The velocity constraints are determined in an analogous, iterative way.

2.2. *MILC SHAKE*

An alternative algorithm for solving for the constraint forces of a system with a linear architecture was first described in Lowe and de Leeuw [10]. As with SHAKE, the approach involves first linearizing the constraint equations, by approximating the terms of order λ^2 as zero. Restricting our attention to a system that is linear, Eq. (7) reduces to

$$\begin{aligned}&\sigma_{i,i+1}(\{\mathbf{r}(t+\Delta)\}) = l_{i,i+1}^2 - \tilde{\mathbf{r}}_{i,i+1}{}^2(t+\Delta) =\\ &\quad 2\,\tilde{\mathbf{r}}_{i,i+1}(t+\Delta)\cdot\left[-\frac{\lambda_{i,i-1}}{m_i}\mathbf{r}_{i-1,i}(t) + \frac{\lambda_{i,i+1}}{\mu_{i,i+1}}\mathbf{r}_{i,i+1}(t) - \frac{\lambda_{i+1,i+2}}{m_{i+1}}\mathbf{r}_{i+1,i+2}(t)\right]\end{aligned} \qquad (10)$$

where μ_{ij} is the reduced mass of two sites i and j defined as $\mu_{ij} = m_i m_j/(m_i + m_j)$. The matrix form of this set of equations – when they are ordered sequentially, matching

their relative position along the contour of the chain – is of a very special form: it is tridiagonal. Inverting a tridiagonal matrix can be done easily and efficiently in order n operations.

In matrix form, we are solving the set of equations $\sigma = \mathbf{J}\lambda$. The vector σ is the set of n constraint conditions at $(t + \Delta)$, the left hand side of Eq. (10), λ is the vector of n Lagrange multipliers, and $\mathbf{J}$ is the $n \times n$ coefficient matrix, where the lower diagonal, diagonal, and upper diagonal entries are, respectively,

$$\begin{aligned} J_{i,i-1} &= \frac{-2}{m_i}\ \tilde{\mathbf{r}}_{i,i+1}(t+\Delta)\cdot\mathbf{r}_{i-1,i}(t), \\ J_{i,i} &= \frac{2}{\mu_{i,i+1}}\ \tilde{\mathbf{r}}_{i,i+1}(t+\Delta)\cdot\mathbf{r}_{i,i+1}(t), \\ J_{i,i+1} &= \frac{-2}{m_{i+1}}\ \tilde{\mathbf{r}}_{i,i+1}(t+\Delta)\cdot\mathbf{r}_{i+1,i+2}(t). \end{aligned} \tag{11}$$

The first and last equations representing the end constraints only have a diagonal and one off-diagonal component. Explicitly, they are

$$\begin{aligned} J_{1,1} &= \frac{2}{\mu_{1,2}}\ \tilde{\mathbf{r}}_{1,2}(t+\Delta)\cdot\mathbf{r}_{1,2}(t), \\ J_{1,2} &= \frac{-2}{m_2}\ \tilde{\mathbf{r}}_{1,2}(t+\Delta)\cdot\mathbf{r}_{2,3}(t), \\ J_{n,n-1} &= \frac{-2}{m_n}\ \tilde{\mathbf{r}}_{n,n+1}(t+\Delta)\cdot\mathbf{r}_{n-1,n}(t), \\ J_{n,n} &= \frac{2}{\mu_{n,n+1}}\ \tilde{\mathbf{r}}_{n,n+1}(t+\Delta)\cdot\mathbf{r}_{n,n+1}(t). \end{aligned} \tag{12}$$

Due to the linearization, the set of equations is not exact. An iterative procedure is still required to converge to the correct values of the forces. We can identify the matrix $\mathbf{J}$ as the Jacobian of the vector of linearized constraint equations and use a simplified Newton method, such as the chord method [8,16], to solve for $\boldsymbol{\lambda}$. On the first iteration, the matrix equation is solved "as is" for a first approximation. A correction term $\boldsymbol{\delta}$ is added to the right hand side at each iteration. This term approximates the non-linear contribution and tends to zero in subsequent iterations, requisite for convergence.

$$\mathbf{J}\ \boldsymbol{\lambda}^k = \boldsymbol{\sigma}^k = \boldsymbol{\sigma}^{k-1} + \boldsymbol{\delta}^k \tag{13}$$

$$\delta^k_{i,i+1} = l^2_{i,i+1} - \mathbf{r}^k_{i,i+1}(t+\Delta)^2 \tag{14}$$

The superscript k indexes the iteration. The set of vectors, $\mathbf{r}^k_{i,i+1}(t+\Delta)$, are the updated link vectors calculated at iteration k using the latest values of the constraint forces to update the site positions. The Jacobian is calculated only once. Although iteration is still required, MILC SHAKE retains the coupling present between constraints, hopefully leading to a more accurate calculation of the forces, necessitating fewer iterations.

2.3. *Velocity constraints*

In order to integrate the equations of motion using velocity Verlet, as described by Andersen [5], one also needs to enforce the time derivative of the constraint equations.

$$\mathbf{v}_{ij}(t+\Delta)\cdot\mathbf{r}_{ij}(t+\Delta) = 0 \tag{15}$$

The vector $\mathbf{v}_{ij}$ is the relative velocity between sites i and j, defined $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$. The updated velocities are

$$\mathbf{v}_i(t+\Delta) = \tilde{\mathbf{v}}_i(t+\Delta) + \frac{\Delta}{2m_i}\left(\mathbf{F}_i^C(t) + \mathbf{F}_i^{CV}(t)\right), \tag{16}$$

where $\mathbf{F}^C$ are the position constraints calculated above. The remaining contribution, labeled $\mathbf{F}^{CV}$, is the constraint force used to update the velocities:

$$\mathbf{F}_i^{CV} = \frac{2}{\Delta}\sum_{i\to p}\lambda_{ip}^V \mathbf{r}_{ip}(t+\Delta). \tag{17}$$

The λ^V's are the undetermined Lagrange multipliers for the velocity constraints. Written explicitly, the updated velocities at $(t+\Delta)$ are

$$\mathbf{v}_i(t+\Delta) = \mathbf{v}_i^0(t+\Delta) + \frac{1}{m_i}\sum_{i\to p}\lambda_{ip}^V \mathbf{r}_{ip}(t+\Delta), \tag{18}$$

where

$$\mathbf{v}_i^0(t+\Delta) = \tilde{\mathbf{v}}_i(t+\Delta) + \frac{\Delta}{2\ m_i}\mathbf{F}_i^C. \tag{19}$$

Substituting Eq. (18) into Eq. (15), we arrive at the set of equation we need to solve:

$$-\mathbf{v}_{ij}^0(t+\Delta)\cdot\mathbf{r}_{ij}(t+\Delta) = \left(\frac{1}{m_i}\sum_{i\to p}\lambda_{ip}^V\mathbf{r}_{ip}(t+\Delta) - \frac{1}{m_j}\sum_{j\to q}\lambda_{ij}^V\mathbf{r}_{jq}(t+\Delta)\right)\cdot\mathbf{r}_{ij}(t+\Delta). \tag{20}$$

For a linear chain, this reduces to,

$$-\mathbf{v}_{i,i+1}^0\cdot\mathbf{r}_{i,i+1} = \left(-\frac{1}{m_i}\lambda_{i-1,i}^V\mathbf{r}_{i-1,i} + \frac{1}{\mu_{i,i+1}}\lambda_{i,i+1}^V\mathbf{r}_{i,i+1} - \frac{1}{m_{i+1}}\lambda_{i+1,i+2}^V\mathbf{r}_{i+1,i+2}\right)\cdot\mathbf{r}_{i,i+1}. \tag{21}$$

All values are taken at $(t+\Delta)$, which has been omitted for clarity. The first and last equations for the ends only have a diagonal and one off-diagonal component, analogous to the position constraints.

One will first notice that there are no non-linear terms. Additionally, this set of equations is tridiagonal. Using MILC SHAKE, the calculation of the Lagrange multipliers for the velocity constraints is a non-iterative procedure consisting of one inversion of a tridiagonal matrix. By comparison, RATTLE uses the same iterative method as that used for the positions, which turns out to be as computationally expensive. In the results section, the performance of the two algorithms is compared.

3. Results

To compare the efficiency of MILC SHAKE compared to SHAKE/RATTLE, a series of simulations was carried out. All calculations were performed on a desktop workstation with an Intel Pentium IV processor (2.6 GHz). For comparative purposes, SHAKE/RATTLE library code from the Collaborative Computational Projects [17] was used.

We begin with an example system of a single linear chain of $N = 100$ connected sites. Each pair of neighboring sites is constrained to be at a separation $l = 1$. For a linear geometry, $n = N - 1$. The starting configuration is a helical shape with a radius of five and a pitch of two. The initial velocity was set to zero.

The first case we consider is a full dynamic simulation of the chain evolving under the influence of a three body potential. A very simple bending energy was used with the form

$$U_{elast.} = \frac{A\ l}{2} \sum \theta_i^2. \tag{22}$$

This potential is simple but not irrelevant. It can be used to model an elastic filament [13,18–20] in which case the constant A is the discrete flexure when l is treated as a segment length. The variable θ_i is the angular deviation from a straight line of adjacent segments meeting at the i^{th} site, defined by

$$cos\left(\theta_i\right) = \frac{1}{l_{i,i+1} l_{i-1,i}} \left(\mathbf{r}_{i+1} - \mathbf{r}_i\right) \cdot \left(\mathbf{r}_i - \mathbf{r}_{i-1}\right). \tag{23}$$

Eq. (2), with $U_{elast.}(t)$ as $U(t)$, is integrated using velocity Verlet [21]. The expected behavior of the helix when evolved for a substantial duration of time is oscillatory. With only one turn of the helix, the system will unfurl to reach a potential energy minimum and then subsequently wrap into a helical form of opposite chirality, all the while conserving the sum of potential and kinetic energy. At this local kinetic energy minimum, the shape is close but not quite an exact inverse of the starting configuration. Over long periods of time, other vibrational modes become excited, leaving no memory of the initial configuration.

We define the error at any time t as

$$\frac{|r_{ij}(t) - l_{ij}|}{l_{ij}} \leq \tau,\ \forall\ ij. \tag{24}$$

The predefined value of τ, also referred to here as the "accuracy", is the maximum allowable violation of any one constraint. Acceptable accuracies range from 10^{-6} down to machine accuracy, depending on the specifics of the system. The time step was chosen such that the error in the position constraints prior to applying constraint forces equaled roughly 0.1%. This is comparable to the starting error typically encountered in a molecular dynamics simulation.

The first test was to determine the number of iterations required to achieve a specified error in the new positions (velocity constraints are added below). Results are plotted in Fig. 1. A moderate accuracy of 10^{-6} in the position constraints is typically achieved in one iteration using MILC SHAKE. In contrast, the same error using SHAKE is attained after nearly a hundred times more iterations. This would be irrelevant if the computational overhead per iteration was significantly higher for MILC SHAKE, but this is not the case. Fig. 2 shows the CPU time needed to achieve a specified τ, plotted on a logarithmic scale. The graphical form of the computational cost scales similarly with the number of iterations for both methods. Therefore, the CPU time required for one iteration is very similar. Consequently, by this measure (the most important measure from a practical point of view) the relative efficiency of MILC SHAKE to SHAKE is the same as one would conclude from the stand point of the number of iterations required. Namely, for moderate accuracy MILC SHAKE is at least two order orders of magnitude better. As Fig. 2 also shows, the CPU time required to achieve a smaller error increases more rapidly

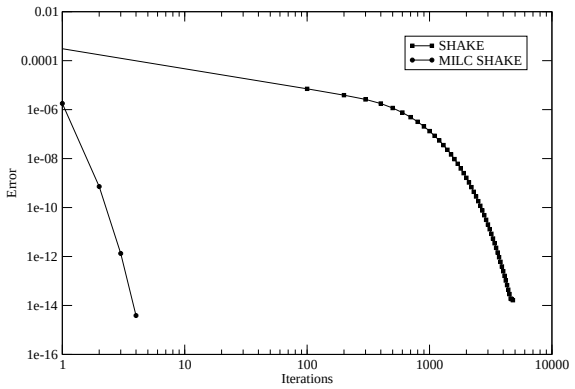

Fig. 1. Error as a function of the number of iterations to calculate the position constraints using SHAKE and MILC SHAKE for a dynamic system of 100 sites.

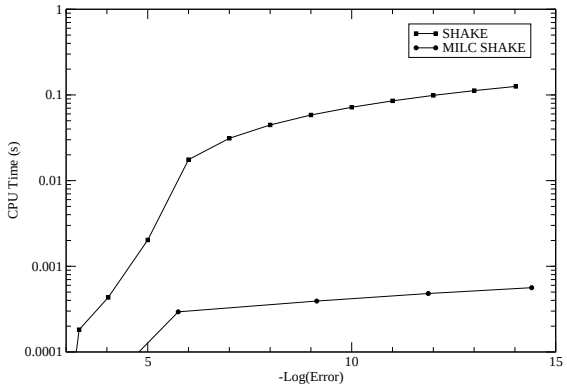

Fig. 2. CPU time as a function of error required to calculate the position constraints using SHAKE and MILC SHAKE for a dynamic system of 100 sites.

for SHAKE than MILC SHAKE. This means that the latter outperforms the former by an increasing margin the higher the accuracy required.

We now turn our attention to the case where we satisfy the constraints on both the site separations and relative velocities. Our point of comparison is therefore RATTLE. Using the iterative procedure RATTLE to calculate the velocity constraints requires a number of iterations broadly equivalent to the number required for the positional constraints. Furthermore, the computational cost per iteration is equivalent. Consequently, RATTLE has twice the complexity of SHAKE and will take twice as long. In contrast, using MILC SHAKE only one tridiagonal matrix inversion is required to calculate the velocity constraints. For this particular system, the calculation time was measured to be 45 μs to achieve an accuracy of better than 10^{-13} (the accuracy is only limited by machine precision). This is relatively short compared to the time required to solve for the positional constraints. That is, for MILC SHAKE there is a negligible additional expense to apply constraints to the relative velocities.

In Fig. 3 we show the overall performance of RATTLE compared with MILC SHAKE. The total CPU time as a function of accuracy (in both the positional and velocity con-

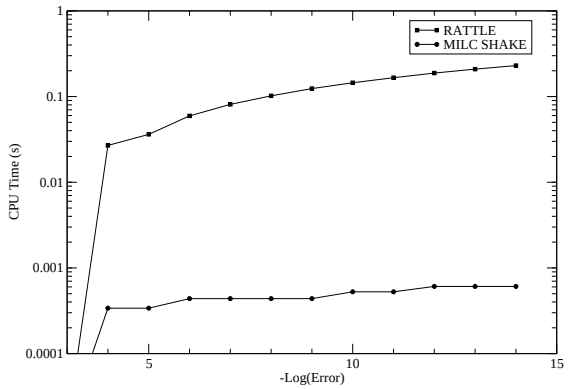

Fig. 3. Combined CPU time required to calculate both the position and velocity constraints using RATTLE and MILC SHAKE for a system of 100 sites.

straints) is, to a good approximation, twice as high for RATTLE compared to the SHAKE results. In contrast, for MILC SHAKE it is essentially the same. Since the scaling is relatively unaffected we can conclude that MILC SHAKE outperforms RATTLE by a factor of two better than it outperforms SHAKE (a rather paltry saving compared to the two orders of magnitude, but still significant).

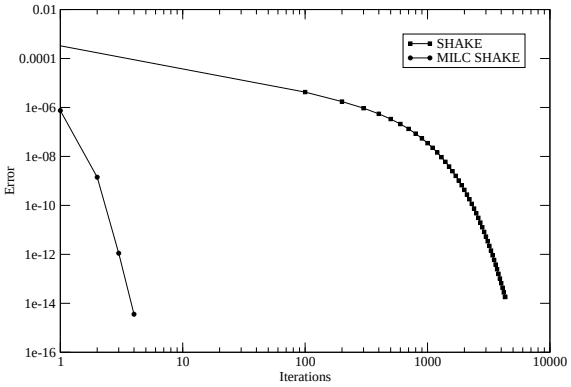

Fig. 4. Error as a function of the number of iterations to calculate the position constraints using SHAKE and MILC SHAKE for a system of 100 sites after randomly perturbing the original, constrained configuration.

For the case described above, it is possible that our algorithm outpaces SHAKE and RATTLE to such a great extent because of the manner in which the constraints are violated. That is, the elastic bending potential imposes an energetic incentive for consecutive link vectors to be parallel, causing the constraints to be breached in a correlated fashion. These correlations could render SHAKE/RATTLE more slowly convergent than might otherwise be the case. To test this proposition, we consider the same helical chain of 100 connected sites introduced in the previous example. The elastic bending potential is excluded, and instead the location of each site is perturbed randomly through the addition of Gaussian white noise [22]. The strength of the perturbation is tuned so that the new, unconstrained configuration has an initial error of approximately 0.1%,

the same initial error that was used above. This model more realistically mimics the situation when, for example, a molecule is agitated stochastically by surrounding solvent or its neighbors. The number of iterations required to achieve a desired accuracy in the positional constraints is plotted in Fig. 4. Comparison with Fig. 1 shows that the results are quantitatively the same as for the bending potential model. That is, MILC SHAKE is again at least two orders of magitude faster compared to SHAKE and RATTLE.

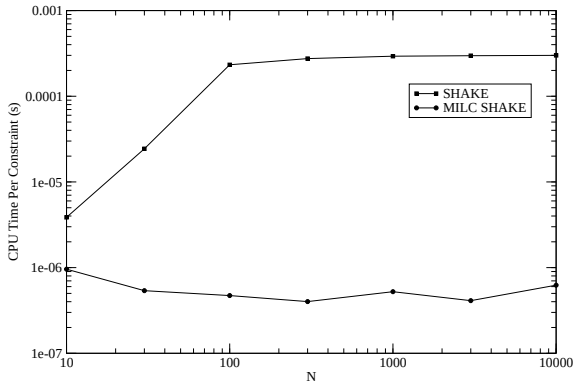

Fig. 5. CPU time per constraint as a function of the system size required to satisfy the position constraints.

Another advantage of this test model is that, because we do not compute the full dynamics, it is simple to consider systems with a much larger numbers of constrained sites (whereas thus far we have only considered $N = 100$). In Fig. 5 we show the CPU time per constraint required to satisfy the site constraints, averaged over 100 independent random violations. System sizes between $N = 10$ and $N = 10,000$ are listed. The maximum tolerated relative error in this case was set to a modest accuracy of 10^{-8}. For $N = 10$, the improvement using MILC SHAKE is a significant factor of three. At larger system sizes, however, the most dramatic improvement is observed. MILC SHAKE is between $100 - 1000$ times faster than SHAKE for system sizes greater than $N = 100$.

Another observation from Fig. 5 is that the CPU time per constraint using MILC SHAKE decreases slightly with N. The decrease in required CPU time per constraint is a reflection of the initial time investment leading to the calculation of the Jacobian. The time it takes to calculate $\mathbf{J}$ becomes comparable to the time of the remaining iterative procedure at smaller N. The reverse trend is present in SHAKE and can be explained generally by noting that with increasing N, there are more degrees of freedom to coordinate, rendering the neglect of coupling increasingly significant with greater system size.

4. Discussion and Conclusions

We described an algorithm MILC SHAKE that solves the problem of imposing rigid constraints in a dynamic simulation of a linear chain using matrix inversion. It is at least two orders of magnitude faster than SHAKE or RATTLE for larger systems, and a significant factor of three faster for the smallest number of sites that we investigated ($N = 10$). This was true for both test cases so we believe this conclusion is system

independent. A pseudo code implementing MILC SHAKE is included as an appendix. Code in FORTRAN 90 is also available for download [23]. Clearly, the only time it would make sense to prefer SHAKE or RATTLE for this problem would be if satisfying the constraints took a negligible proportion of the overall CPU time per time step. This in turn depends on the degree of accuracy required. MILC SHAKE is particularly advantageous where high accuracy is required. Why might this ever be necessary? For a given time step, the degree of violation of the constraints determines how much the total (bending plus kinetic) energy drifts during a dynamic simulation [7]. In the absence of integration errors it should of course be a constant. Taking as an example the system with the bending potential described above, a tolerance of at least 10^{-8} is needed to keep the relative drift in the total energy to within one part in 10^{-4}. This is the error typically tolerated in molecular dynamics calculations. Quantitatively this might be system dependent. Nonetheless, with MILC SHAKE it is practical, if needed, to reach machine accuracy at a modest computational cost and eliminate any such problem.

The number of iterations SHAKE requires to converge depends on the degree to which the constraints are violated in one time step. The smaller this violation, the fewer iterations SHAKE requires. The same applies to MILC SHAKE. For more complicated systems than the one considered here, the requirement that the time step is short enough to adequately integrate the unconstrained equations might result in fewer iterations. This was indeed the case for the systems considered by Barth *et al.* It is also possible that the library SHAKE routine that we compare against is not optimal, in that it requires more iterations than necessary. However, given that the CPU time for one iteration of MILC SHAKE is similar to that of SHAKE, MILC SHAKE is significantly more efficient so long as SHAKE requires more iterations. This will generally be the case.

The results we reported here are restricted to the linear topography but it is imporatant to stress that a ring topology is a trivial extension. All one has to do is replace the tridiagonal solver with a cyclic tridiagonal matrix solver [11] or manipulate the matrix to tridiagonal form [10]. The latter is easily done in order n operations. The overhead associated with this procedure only increases the CPU time per iteration by a factor of approximately two. The rate of convergence is relatively insensitive to the change in topology. This is in stark contrast to SHAKE and RATTLE which we found converge much more slowly for this topology and often do not converge at all [14]. That is why we do not provide a comparison here. Suffice to say that if MILC SHAKE is preferable for the linear topology it is infinitely preferable for the ring topology.

The method is not, of course, completely general; however, almost all conceivable problems can be broken down into at least parts that could be solved in isolation using MILC SHAKE. For example, in a benzene ring the ring constraints could be treated independently so that only the contraints for the hydrogens are unknown (of course the two sets of constraints are also coupled). Similarly, for a branched linear system each branch could be solved independently and only the contstraint force at the points where they meet would need calculating. The introduction of bifurcations leads to a matrix which is still predominantly tridiagonal and sparse, but that contains some non-tridiagonal elements. Possibly a scheme could be devised to make use of the specific form of the matrix? Alternatively, a hybrid scheme iterating between using MILC SHAKE for groups of linked elements and using a SHAKE-like calculation at the joints offers the prospect of a generalized scheme.

A.G.B thanks the Thouron Award and the National Science Foundation Graduate

Research Fellowship Program for funding.

Appendix A. Pseudo Fortran Code

The following code calculates the constrained positions and velocities of a linear, open chain after a single time step. The equation of motion is integrated using velocity Verlet, and all constraints are calculated using MILC SHAKE. Two subroutines are included. The first (MILCSHAKEa) calculates the new, constrained positions and updates the velocities a half step; the second (MILCSHAKEb) finishes the calculation of the constrained velocities, using the forces calculated with the new, constrained positions. The complete version of the code is available for download [23].

```
#   VARIABLES
# NS,NL             Number of sites, links
# TOL               The predefined error
# M                 (NS) Masses of the sites
# DSQ               (NL) Squared link lengths
# DT                Time step
# F,V,R             (3,NS) Forces, velocities, positions
# DONE              Indicates when the tolerance has been achieved
# A,B,IT            Indices
# ERROR             Calculated error
# MAX               Maximum error
# RIJ               (3,NL) Link vectors using R, defined R(I)-R(J)
# P                 (3,NS) Updated positions prior to constraints
# RNCIJ             (3,NL) Link vectors using P
# N                 (3,NS) Current approximation to constrained positions
# NIJ               (3,NL) Link vectors using N
# RIJ2,RNCIJ2,NIJ2 (NL) Link vectors squared
# DL,DU             (NL-1) Subdiagonal and superdiagonal vectors of the Jacobian
# D                 (NL) Diagonal vector of the Jacobian
# LAMBDA            (NL) Undetermined Lagrange multipliers
# RHS,RHSOLD        (NL) Constraint conditions
#   FUNCTIONS
# SQRT(...)         Calculates the square root
# DOT(VECT1,VECT2) Calculates the inner product of VECT1 and VECT2
# MU(M1,M2)         Calculates the reduced mass of M1 and M2
# TRIDIAG(...)      Solves the tridiagonal matrix

SUBROUTINE MILCSHAKEa ( R, V, F, DT, DSQ, M, TOL, NS, NL )
#   1st step of velocity Verlet integration
  DO A = 1,NS
    P(:,A) =  R(:,A) + DT * V(:,A) + DT * DT / 2.0 * F(:,A) / M(A)
    V(:,A) =  V(:,A) + DT / 2.0 * F(:,A) / M(A)
  ENDDO
#   Set up of matrix equation
```

```
  DO A = 1,NL
    B = A+1
    RIJ(:,A)    = R(:,A)-R(:,B)
    RIJ2(A)     = DOT(R(:,A),R(:,A))
    RNCIJ(:,A)  = P(:,A)-P(:,B)
    RNCIJ2(A)   = DOT(RNCIJ(:,A),RNCIJ(:,A))
  ENDDO
  D(1)    =  2.0*( DOT(RIJ(:,1),RNCIJ(:,1)) )/ MU(M(1),M(2))
  DU(1)   = -2.0*( DOT(RIJ(:,2),RNCIJ(:,1)) )/ M(2)
  DO A = 2,NL-1
    DL(A-1)= -2.0*( DOT(RIJ(:,A-1),RNCIJ(:,A)) )/ M(A)
    D(A)   =  2.0*( DOT(RIJ(:,A),RNCIJ(:,A))   )/ MU(M(A),M(A+1))
    DU(A)  = -2.0*( DOT(RIJ(:,A+1),RNCIJ(:,A)  )/ M(A+1)
  ENDDO
  DL(NL-1)= -2.0*( DOT(RIJ(:,NL-2),RNCIJ(:,NL-1)  )/ M(NL)
  D(NL)   =  2.0*( DOT(RIJ(:,NL),RNCIJ(:,NL)      )/ MU(M(NL),M(NL+1))
  RHS(:) = DSQ(:) - RNCIJ2(:)
  RHSOLD(:) = RHS(:)
#   Iterative solution
  DONE = .FALSE.
  IT = 1
5 IF (.NOT. DONE) THEN
    CALL TRIDIAG(NL,DL,D,DU,RHS,LAMBDA)
    N(:,1) = P(:,1) + LAMBDA(1)*RIJ(:,1)/M(1)
    DO A = 2,NS-1
      N(:,A) = P(:,A) - LAMBDA(A-1)*RIJ(:,A-1)/M(A) +
                LAMBDA(A)*RIJ(:,A)/M(A)
    ENDDO
    N(:,NS) = P(:,NS) - LAMBDA(NS-1)*RIJ(:,NS-1)/M(NS)
    MAX = 0.0
    DO A = 1,NL
      B = A+1
      NIJ(:,A)  = N(:,A)-N(:,B)
      NIJ2(A)   = DOT(NIJ(:,A),NIJ(:,A))
      ERROR     = ABS( SQRT(NIJ2(A)) - SQRT(DSQ(A)) )/ SQRT(DSQ(A))
      IF (ERROR .GT. MAX) THEN
        MAX = ERROR
      ENDIF
    ENDDO
    IF (MAX .LE. TOL) THEN
      DONE = .TRUE.
      V(:,1) = V(:,1) + LAMBDA(1)*RIJ(:,1)/DT/M(1)
      DO A = 2,NS-1
        V(:,A) = V(:,A) + LAMBDA(A)*RIJ(:,A)/DT/M(A) -
                  LAMBDA(A-1)*RIJ(:,A-1)/DT/M(A)
      ENDDO
      V(:,NS) = V(:,NS) - LAMBDA(NS-1)*RIJ(:,NS-1)/DT/M(NS)
```

```
      DO A = 1,NS
        R(:,A) = N(:,A)
      ENDDO
    ELSE
      RHS(:)    = DSQ(:) - NIJ2(:) + RHSOLD(:)
      RHSOLD(:) = RHS(:)
      IT = IT + 1
      GOTO 5
    ENDIF
  ENDIF
END SUBROUTINE MILCSHAKEa


SUBROUTINE MILCSHAKEb ( R, V, F, DT, M,  NS, NL )
#   2nd step of velocity Verlet integration
  DO A = 1, NS
    V(:,A) = V(:,A) + DT / 2.0 * F(:,A) / M(A)
  ENDDO
#   Set up and solution of matrix equation
  DO A = 1, NL
    B = A+1
    VIJ(:,A) = V(:,A) - V(:,B)
    RIJ(:,A) = R(:,A) - R(:,B)
    RHS(A) = - ( DOT(VIJ(:,A),RIJ(:,A)) )
  ENDDO
  D(1) =   ( DOT(RIJ(:,1),RIJ(:,1)) )/ MU(M(1),M(2))
  DU(1)= - ( DOT(RIJ(:,2),RIJ(:,1)) )/ M(2)
  DO A = 2,NL-1
    DL(A-1) = - ( DOT(RIJ(:,A-1),RIJ(:,A)) )/ M(A)
    D(A)    =   ( DOT(RIJ(:,A),RIJ(:,A))   )/ MU(M(A),M(A+1))
    DU(A)   = - ( DOT(RIJ(:,A+1),RIJ(:,A)) )/ M(A+1)
  ENDDO
  DL(NL-1) = -  ( DOT(RIJ(:,NL-1),RIJ(:,NL)) )/ M(NL)
  D(NL)    =    ( DOT(RIJ(:,NL),RIJ(:,NL))   )/ MU(M(NL),M(NL+1))
  CALL TRIDIAG(NL,DL,D,DU,RHS,LAMBDA)
  V(:,1) = V(:,1) + LAMBDA(1)*RIJ(:,1)/ M(1)
  DO A = 2,NS-1
    V(:,A) = V(:,A) + ( LAMBDA(A)*RIJ(:,A) - LAMBDA(A-1)*RIJ(:,A-1) )/ M(A)
  ENDDO
  V(:,NS) = V(:,NS) - LAMBDA(NS-1)*RIJ(:,NS-1)/ M(NS)
END SUBROUTINE MILCSHAKEb
```

References

[1] T. Kastenmeier and F.J. Vesely, Robotica 14, (1996) 329.

[2] J.M. Zhao, N.I. Badler, Inverse kinematics positioning using nonlinear-programming for highly articulated figures, ACM Transactions on Graphics, 13 (4) (1994) 313-336.

[3] Virtual Reality Software and Technology archive, Proceedings of the ACM symposium on Virtual reality software and technology table of contents, Taipei, Taiwan, Pages: 153 - 162, Year of Publication: 1998, ISBN:1-58113-019-8.

[4] J.-P. Ryckaert, G. Ciccotti, H.J.C. Berendsen, Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes, J. Comput. Phys. 23 (1977) 327-341.

[5] H. Andersen, RATTLE: a velocity version of the shake algorithm for molecular-dynamics calculations, J. Comput. Phys. 52 (1) (1983) 24-34.

[6] S. Miyamoto, P.A. Kollman, SETTLE: An analytical version of the SHAKE and RATTLE algorithm for rigid water models, J. Comput. Chem. 13 (8) (1992) 952-962.

[7] V. Kräutler, W.F. van Gunsteren, P.H. Hünenberger, A fast SHAKE algorithm to solve distance constraint equations for small molecules in molecular dynamics simulations, J. Comput. Chem. 22 (5) (2001) 501-508.

[8] P. Gonnet, P-SHAKE: A quadratically convergent SHAKE in O(n), J. Comput. Phys. 220 (2007) 740-750.

[9] E. Barth, K. Kuczera, B. Leimkuhler, R.D. Skeel, Algorithms for constrained molecular dynamics, J. Comput. Chem. 16 (1995) 1192.

[10] C.P. Lowe, S.W. de Leeuw, Strategies for constraint dynamics, Proc. 5th Ann. Conf. Advanced School for Computing and Imaging (ed. A. Hoekstra & J.F.M. Tonino) (1999) 279-287. Delft, The Netherlands: Advanced School for Computing and Imaging.

[11] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical recipes in FORTRAN (2nd ed.): the art of scientific computing, Cambridge University Press, New York, NY, 1992.

[12] J.I. Siepmann, S. Karaborni, B. Smit, Simulating the critical properties of complex fluids, Nature. 365, (1993) 330-332.

[13] M.C. Lagomarsino, I. Pagonabarraga, C.P. Lowe, Hydrodynamic induced deformation and orientation of a microscopic elastic filament, Phys. Rev. Lett. **94** 148104 (2005).

[14] T.R. Forester, W. Smith, SHAKE, RATTLE, and Roll: Efficient constraint algorithms for linked rigid bodies, J. Chem. Phys. 19 (1), (1997) 102-111.

[15] H. Goldstein, Classical Mechanics (2nd edn). Addison-Wesley, Reading, MA, 1980.

[16] P.E. Gill, W. Murray, M.H. Wright, Numerical linear algebra and optimization, Addison-Wesley, Redwood City, California, 1991.

[17] Collaborative Computational Projects 5 Program Library [http://www.ccp5.ac.uk/librar.shtml].

[18] R.G. Cox, The motion of long slender bodies in a viscous fluid. Part I. General Theory. J. Fluid Mech. 44, 791-810 (1970)

[19] G.K. Batchelor, Slender-body theory for particle of arbitrary cross-section in Stokes flow, J. Fluid Mech. 44, 419-440 (1970)

[20] C.P. Lowe,Dynamics of filaments: modelling the dynamics of driven microfilaments. Phil. Trans. Roy. Soc. London. B 358, 1543-1550 (2003).

[21] M.P. Allen, D.J. Tildesley, Computer Simulation of Liquids. Oxford University Press, Oxford (1989).

[22] W.P. Petersen, Lagged Fibonacci Random Number Generators for the NEC SX-3, International Journal of High Speed Computing (1994).

[23] [http://www.cmth.ph.ic.ac.uk/people/aimee.bailey/milcshake.html] If the webpage no longer exists at the time of attempted access, please contact the authors for code.