Published in final edited form as:

Eur Phys J Spec Top. 2011 November 1; 200(1): 211–223. doi:10.1140/epjst/e2011-01525-9.

SHAKE parallelization

Ron Elber¹, A. Peter Ruymgaart¹, and Berk Hess²

¹ Institute for Computational Engineering and Sciences, Department of Chemistry and Biochemistry, University of Texas at Austin, TX 78712, USA

² Royal Institute of Technology (KTH), Department of Theoretical Physics, Albanova University Center, SE-10691 Stockholm, Sweden

Abstract

SHAKE is a widely used algorithm to impose general holonomic constraints during molecular simulations. By imposing constraints on stiff degrees of freedom that require integration with small time steps (without the constraints) we are able to calculate trajectories with time steps larger by approximately a factor of two. The larger time step makes it possible to run longer simulations. Another approach to extend the scope of Molecular Dynamics is parallelization. Parallelization speeds up the calculation of the forces between the atoms and makes it possible to compute longer trajectories with better statistics for thermodynamic and kinetic averages. A combination of SHAKE and parallelism is therefore highly desired. Unfortunately, the most widely used SHAKE algorithm (of bond relaxation) is inappropriate for parallelization and alternatives are needed. The alternatives must minimize communication, lead to good load balancing, and offer significantly better performance than the bond relaxation approach. The algorithm should also scale with the number of processors. We describe the theory behind different implementations of constrained dynamics on parallel systems, and their implementation on common architectures.

I. Introduction

Molecular Dynamics (MD) is a useful tool to test statistical mechanic theories and approximations, predict macroscopic properties of materials, and simulate molecular functions in biophysical systems. The broad applicability of Molecular Dynamics is not without cost. The trajectories are expensive to compute and require significant computational resources and time. It is common to find Molecular Dynamics projects using millions of CPU hours. The calculations are particularly challenging due to the broad range of time scales of molecular systems. Periods of individual bond vibrations start with a few femtoseconds, but relevant time scales of (for example) protein folding, extend to seconds. To retain the stability of the calculations the fast degrees of freedom must be followed accurately using small time steps, which means that 10^{15} time steps are required to reach a second (!), a formidable task. Approaches to speed up the calculations are therefore highly desired.

A useful approach to extend simulation times exploits the combination of MD with constraints. Instead of explicitly integrating fast degrees of freedom (such as bonds) as a function of time, the bond lengths are fixed at ideal values. With the elimination of some fast degrees of freedom larger time steps can be used. Of course, from a statistical mechanics point of view the constrained system is not equivalent to the flexible system we started with. However, since bond displacements are typically small and restrained by harmonic potentials to a single equilibrium value, the constraints affect only mildly the average geometries. Furthermore, the force field can be parameterized to restore lost flexibility due to the additional rigidity of the system [Robert D. Skeel and Sebastian Reich,

“Corrected potentials for constrained dynamics”, this issue]. Angle and bond rigidity can make it more difficult for a polymer to execute large-scale motions that include bond rotations, since the barriers in that case can be significantly higher. For example 1–4 interaction strengths, a widely used term in empirical force fields, can be reduced to facilitate more rapid bond rotations.

Interestingly, the overall gain in time step size due to constraints is modest (rarely more than a factor of 2). One explanation is the existence of other fast degrees of freedom in the form of collisions. In condensed phase we frequently find particles that are too close to each other, executing a hard collision due to more than an average relative velocity. During a collision the particles change courses rapidly; their motions are fast and need to be integrated with small time steps. The difficulty in eliminating this type of motions with constraints is that the collisions are transient and the identities of the colliding particles are constantly changing. Therefore there is an upper bound to the time step that can be used effectively with SHAKE. The speedup factor that we can obtain without significant compromises in energy conservation is about a factor of 2.

It is useful to note that bond angles have similar properties to bonds. They are restrained by stiff harmonic potentials, though less steep than those of bonds, and have a single equilibrium geometry. They are clearly reasonable candidates to be constrained, besides the bonds, to further gain computational efficiency by reducing the number of degrees of freedom. Indeed it is common practice in Monte Carlo studies of proteins to invoke only the torsion degrees of freedom. However the complexity of angle constraints, especially in conjunction with the widely used implementation of SHAKE for bond relaxation, made the constraints of angles harder to implement. It turns out that a particular implementation of the angle constraints (via 1–3 bonds) is actually convenient for parallelization and is discussed in the present manuscript.

The manuscript is organized as follows: in the next section **Algorithm** we describe the general formulation of SHAKE. In **Parallelism** we discuss different strategies to make SHAKE run in parallel and we conclude with some future ideas.

II. Algorithm

The formulation below is based, of course, on the seminal paper introducing SHAKE [1]. It also follows closely the ideas outlined in Barth et al. [2], Weinbach and Elber [3] and Hess [4, 5].

The constraints are introduced to the equations of motions using the formulation of Lagrange multipliers. The usual equations of motions take the form

$$M \frac{d^2 X}{dt^2} = -\nabla U - \sum_{\alpha} \lambda_{\alpha} \nabla \sigma_{\alpha} \quad (1)$$

where M is the diagonal mass matrix, X the Cartesian coordinate vector of all the particles in the system, t the time, and U is the potential energy. The coefficients λ_{α} are Lagrange multipliers and σ_{α} are constraints, typically holonomic.

$$\sigma_{\alpha}(X) = 0 \quad (2)$$

Lagrange multipliers are determined with the help of the constraints' equations. The algorithm of SHAKE [1] is based on the observation that the constraints must be satisfied

exactly, i.e., each time step. Otherwise a drift in the constraint values will be obtained during the calculations, which will accumulate to unacceptable errors. By “exact” we mean that the algorithm is exact and the only source of errors in the calculation are those of computer floating point truncation. We call SHAKE, an algorithm that satisfies the arbitrary constraint equations within a pre-specified error.

It is useful to illustrate the way SHAKE works in the context of a popular specific algorithm and one type of constraints. We therefore consider the Verlet algorithm and distance constraints. The distance constraints are formulated as:

$$\sigma_{\alpha \equiv (jk)}(X) = r_{jk}^2 - r_{jk,0}^2 = 0 \quad \forall \alpha = 1, \dots, L \quad (3)$$

where r_{jk}^2 is the square of the distance between particles j and k that are kept constrained, $r_{jk,0}^2$ is the ideal value to which the square of the distance is compared to, L is the number of constrained distances. To keep the equations compact we frequently use the index α to denote a constraint and not the indices of the pair of the particles.

Returning the discussion to a general formulation of constraints, the velocity Verlet algorithm with constraints is

$$X_{i+1} = X_i + V_i \Delta t - \frac{\Delta t^2}{2} M^{-1} \left(\nabla U(X_i) + \sum_{\alpha} \lambda_{\alpha,i} \nabla \sigma_{\alpha}(X_i) \right) \quad (4)$$

The index i is for the time, and Δt is the time step. It is useful to first write the expression for general constraints and then migrate to the special case of distance constraints. Equation (2) implies that the constraints must be satisfied exactly at all times and specifically also at time index $i + 1$. We use the condition, $\sigma(X_{i+1}) = 0$ to derive equations for the Lagrange multipliers.

We define for convenience a new coordinate set, constructed without constraints:

$X_i^{(0)} = X_i + V_i \Delta t - \frac{\Delta t^2}{2} M^{-1} \nabla U(X_i)$ which is used to derive an explicit equation for the Lagrange multipliers. The superscript on the coordinate index is the number of SHAKE iterations, which for the move without constraints is zero.

$$\sigma_{\alpha}(X_{i+1}) = \sigma_{\alpha} \left(X_i^{(0)} - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta} \nabla \sigma_{\beta}(X_i) \right) = 0 \quad (5)$$

The significant breakthrough of the SHAKE algorithm was to realize that the constraints must be solved exactly at each step to avoid accumulations of errors. This is achieved by iterations of a linearized problem. Expanding the constraint near $X_i^{(0)}$ we have

$$\sigma_{\alpha}(X_{i+1}) \cong \sigma_{\alpha}(X_i^{(0)}) - \nabla \sigma_{\alpha}(X_i^{(0)}) \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta} \nabla \sigma_{\beta}(X_i) = 0 \quad (6)$$

Equation (6) is a linear approximation for a nonlinear equation for the constraints. An exact solution for equation (6) will provide only an approximate solution to the constraint

equations, and we therefore must iterate the solution in the spirit of Newton's solution to non-linear problems. Define the following matrix

$$A_{\alpha\beta}^{(0)} = \nabla\sigma_{\alpha}(X_i^{(0)}) \frac{\Delta t^2}{2} M^{-1} \nabla\sigma_{\beta}(X_i) \quad (7)$$

Eq. (7) defines an asymmetric matrix. For computational convenience in some algorithms it is useful to define a symmetric matrix $A_{\alpha\beta} = \nabla\sigma_{\alpha}(X_i) \frac{\Delta t^2}{2} \nabla\sigma_{\beta}(X_i)$. It is an approximation of order of Δt^2 to $A_{\alpha\beta}^{(0)}$ that will be considered in more detail in section **III.3.1**. Different approximate forms of the matrix were discussed extensively in [2] and [3] and were revisited more recently in [6–8]. The last equation provides a more compact expression for determining the Lagrange multipliers

$$\sum_{\beta} A_{\alpha\beta}^{(0)} \lambda_{\beta} = \sigma_{\alpha}(X_i^{(0)}) \quad (8)$$

Let the solution of the linear equation be $\lambda_{\beta}^{(0)}$. Because of the linearization of the general constraint equation (Eq. (5)) the solution is approximate. An approximate solution is still useful and we can define an adjusted coordinate set as

$$X_i^{(1)} = X_i^{(0)} - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta}^{(0)} \nabla\sigma_{\beta}(X_i) \quad (9)$$

The new coordinate is used in another (linear) solution of modified Lagrange multipliers. Instead of equation (5), we write

$$\begin{aligned} \sigma_{\alpha}(X_{i+1}) &= \sigma_{\alpha}\left(X_i^{(0)} - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta}^{(0)} \nabla\sigma_{\beta}(X_i)\right) \\ &= \sigma_{\alpha}\left(X_i^{(1)} + \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta}^{(0)} \nabla\sigma_{\beta}(X_i) - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta}^{(0)} \nabla\sigma_{\beta}(X_i)\right) \\ &= \sigma_{\alpha}\left(X_i^{(1)} - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} (\lambda_{\beta} - \lambda_{\beta}^{(0)}) \nabla\sigma_{\beta}(X_i)\right) = 0 \end{aligned} \quad (10)$$

If we calculate the first iteration correctly then $|\lambda_{\beta} - \lambda_{\beta}^{(0)}| < |\lambda_{\beta}|$ and the next order iteration is smaller. Expanding (again) the constraints in a small displacement, we have

$$\sigma_{\alpha}(X_{i+1}) \cong \sigma_{\alpha}(X_i^{(1)}) - \nabla\sigma_{\alpha}(X_i^{(1)}) \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} (\lambda_{\beta} - \lambda_{\beta}^{(0)}) \nabla\sigma_{\beta}(X_i) = 0 \quad (11)$$

We define the parameter $\lambda_{\beta}^{(1)} = (\lambda_{\beta} - \lambda_{\beta}^{(0)})$ which is a solution of the following linear equation

$A_{\alpha\beta}^{(1)} \lambda_{\beta}^{(1)} = \sigma_{\alpha}(X_i^{(1)})$ where $A_{\alpha\beta}^{(1)} = \nabla\sigma_{\alpha}(X_i^{(1)}) \frac{\Delta t^2}{2} M^{-1} \nabla\sigma_{\beta}(X_i)$. The solution of the linear equation makes it possible for us to adjust the coordinate position for the second time.

$$X_i^{(2)} = X_i^{(1)} - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta}^{(1)} \nabla \sigma_{\beta}(X_i) \quad (12)$$

Furthermore, the iteration procedure should be clearer by now and we can write for the general n -th iteration

$$X_i^{(n)} = X_i^{(n-1)} - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta}^{(n-1)} \nabla \sigma_{\beta}(X_i) \quad (13)$$

And the corresponding linear equation to solve

$$\begin{aligned} A_{\alpha\beta}^{(n-1)} &= \nabla \sigma_{\alpha}(X_i^{(n-1)}) \frac{\Delta t^2}{2} M^{-1} \nabla \sigma_{\beta}(X_i) \\ \sum_{\beta} A_{\alpha\beta}^{(n-1)} \lambda_{\beta}^{(n-1)} &= \sigma_{\alpha}(X_i^{(n-1)}) \end{aligned} \quad (14)$$

The algorithm converges or the iterations terminate when the current values of the constraints $\sigma_{\alpha}(X_i^{(n)}) \forall \alpha$ are below an error threshold ε , ($|\sigma_{\alpha}(X_i^{(n)})| \leq \varepsilon$). The full algorithm is sketched below:

SHAKE Algorithm for constrained Molecular Dynamics:

1. Evaluate constraints. If $|\sigma_{\alpha}(X_i^{(n)})| \leq \varepsilon \forall \alpha$, stop.
2. Compute the matrix $A_{\alpha\beta}^{(n)}$ following Eq. (14)
3. Solve the linear system $\sum_{\beta} A_{\alpha\beta}^{(n)} \lambda_{\beta}^{(n)} = \sigma_{\alpha}(X_i^{(n)})$ to determine the Lagrange multipliers $\lambda_{\beta}^{(n)}$
4. Compute an adjusted coordinate set: $X_i^{(n+1)} = X_i^{(n)} - \frac{\Delta t^2}{2} M^{-1} \sum_{\beta} \lambda_{\beta}^{(n)} \nabla \sigma_{\beta}(X_i)$
5. Go to 1

Solving the linear equations is probably the most expensive component of the calculations with complexity of $O(L^3)$ in the worst case, where L is the number of constraints. However, in many practical applications the matrix $A_{\alpha\beta}$ is sparse and its diagonal is dominant. In that case it is possible to use a serial algorithm that passes constraint by constraint and adjusts their values. We call this algorithm “bond relaxation”. This algorithm converges rapidly and efficiently for sparse constrained systems [1]. The typical number of iteration is ~50 for biomolecular polymers with a step size of 1fs if the relative accuracy in bond length is set to 10^{-12} for a double precision calculation. It is useful to illustrate this algorithm on the most common application of SHAKE to constrain bond distances (Eq. (3)). We have

$$(\nabla \sigma_{jk}) = 2(r_j - r_k) \quad j \neq k \quad (15)$$

where r_p is the three dimensional vector of atom p . The elements of the matrix $A_{\alpha\beta}^{(n)}$ are now readily computed. Below we have changed the constraint index to reflect more clearly the type of constraint (distance between particles i and j).

$$\begin{aligned}
 A_{jk,pq}^{(n)} &= \frac{\Delta t^2}{2} \nabla \sigma_{jk}(X_i^{(n)}) M^{-1} \nabla \sigma_{pq}(X_i) = \\
 &= 2\Delta t^2 \cdot \sum_{l,l'} (r_j^{(n)} - r_k^{(n)}) (\delta_{ij} - \delta_{lk}) (M^{-1})_{ll'} (r_p - r_q) (\delta_{l'p} - \delta_{l'q}) \\
 &= 2\Delta t^2 \begin{cases} \text{bond is shared } (jk=pq) & r_{jk}^{(n)} \cdot r_{jk} \left(\frac{1}{m_j} + \frac{1}{m_k} \right) \\ \text{atom is shared } (j=p, k \neq q) & r_{jk}^{(n)} \frac{1}{m_j} r_{jq} \\ \text{no sharing} & 0 \end{cases} \quad (16)
 \end{aligned}$$

The diagonal elements of the matrix are larger than any off-diagonal elements and the matrix is sparse. The number of bond constraints is of order N where N is the number of particles and not N^2 the number of matrix elements. This result is encouraging from the perspective of Eq. (14). We are likely to find a solution to an inverse problem of the type

$A_{\alpha\beta} \lambda_\beta = \sigma_\alpha$ as $\lambda_\beta = A_{\beta\alpha}^{-1} \sigma_\alpha$. Both the conjugate gradient approach of Weinbach and Elber [3] and the operator expansion LINCS of Hess [4] are exploiting the “mostly diagonal” character of the matrix, avoiding the explicit calculation of the inverse, even though in different flavors. Parallel solution of the SHAKE algorithm depends on the effective parallel solution of the linear equation and on minimizing the number of iterations which must be done serially.

The constraint equations for the velocities are simple expansion on the discussion above. They are required to complete the algorithm and below we follow the derivation of RATTLE [9]. Since the constraints are constant, so are the time derivatives of the constraints. We have in general for a holonomic constraint

$$\frac{d\sigma_\alpha}{dt} = \nabla \sigma_\alpha \cdot V = 0 \quad \forall \alpha \quad (17)$$

which is a convenient equation to determine the velocities that satisfy the constraints. The equations of motions for the velocities in Velocity Verlet with constraints are:

$$V_{i+1} = V_i - \frac{\Delta t}{2} M^{-1} \left(\nabla U(X_i) + \sum_\alpha \lambda_{\alpha,i} \nabla \sigma_\alpha(X_i) + \nabla U(X_{i+1}) + \sum_\alpha \lambda_{\alpha,i+1} \nabla \sigma_\alpha(X_{i+1}) \right) \quad (18)$$

The Lagrange multiplier $\lambda_{\alpha i}$ were already determined when the corrections to the coordinates were computed. We define

$$V_i^{(0)} \equiv V_i - \frac{\Delta t}{2} M^{-1} \left(\nabla U(X_i) + \sum_\alpha \lambda_{\alpha,i} \nabla \sigma_\alpha(X_i) \right) \quad (19)$$

which we can immediately compute as $(X_{i+1} - X_i)/\Delta t$. In fact it makes sense to SHAKE the difference $(X_{i+1} - X_i)$ instead of the coordinate X_{i+1} to be ready for the calculations of the velocities. We have

$$\begin{aligned}
V_{i+1} \cdot \nabla \sigma_\alpha(X_{i+1}) &= 0 \\
V_i^{(0)} \cdot \nabla \sigma_\alpha(X_{i+1}) - \frac{\Delta t}{2} M^{-1} \left(\nabla U(X_{i+1}) + \sum_\beta \lambda_{\beta,i+1} \nabla \sigma_\beta(X_{i+1}) \right) \cdot \nabla \sigma_\alpha(X_{i+1}) &= 0 \\
\sum_\beta A_{\alpha\beta,i+1} \lambda_{\beta,i+1} &= \Delta t \cdot V_i^{(0)} \cdot \nabla \sigma_\alpha(X_{i+1}) - \frac{\Delta t^2}{2} M^{-1} \nabla U(X_{i+1}) \cdot \nabla \sigma_\alpha(X_{i+1})
\end{aligned} \tag{20}$$

where we recover the expression for the matrix $A_{\alpha\beta}$ for the next step ($i+1$). The right hand side equation is independent of $\lambda_{\beta,i+1}$. The equation for the Lagrange multipliers is now linear and there is no need for iterations. Once $\lambda_{\beta,i+1}$ is determined from Eq. (20) V_{i+1} is readily computed.

III. SHAKE parallelization

III.1 Introduction

Why do we need to parallelize SHAKE at all? This is not a trivial question. In serial calculations the SHAKE algorithm rarely takes more than a few percent of the total calculation time and it seems unnecessary to invest resources to speed up its computing time. However, as Molecular Dynamics codes exploit parallelism and other modern computer architecture like the use of Graphic Processing Unit [10, 11], the clock time of other expensive calculations like the calculations of the non-bonded interactions is much reduced. Implementing a version of the Molecular Dynamics program [12] on the GPU (Ruymgaart, Cardenas and Elber, to be published) we realized that shaking all the bonds in the system (a calculation conducted at present on a single core) requires about 30 percent of the total compute time. To make further progress it is therefore crucial to find a way to parallelize SHAKE as well.

What are the major considerations for effective parallelization of a computer simulation?

First, the job or a significant fraction of it, must be separable to independent tasks to be executed on different cores.

Second, the communication between the cores must be minimized. It turns out that data transfer is very small, and a prime concern is the time to initiate a parallel call, or the so called latency. Hence, it is number of communications that needs to be reduced.

Third, the tasks of the different cores must be balanced. Ideally, no single core should do more work than the others.

Of the three factors mentioned above, the first and the third are more straightforward to handle in the context of an algorithm, while the second depends strongly on the hardware used. Current trends in the industry may eliminate communication concerns all together. Intel recently announced the availability of Knight's Corner, a new chip with 50 cores. The 50 cores share the same memory and therefore remove the communication problem. Of course, if a distributed computing system is the target, the speed and latency of the network can be a bottleneck as was demonstrated in [3].

III.2 "Bond relaxation" and parallelism

There are two main philosophies to conduct SHAKE calculations. The first focuses only on the diagonal elements of the matrix $A_{\alpha\beta}^{(n)} \rightarrow A_{\alpha\alpha}^{(n)}$, and determines approximate Lagrange multiplier using the diagonal elements of the full matrix $\lambda_\alpha^{(n)} \approx \sigma_\alpha(X_i^{(n)})/A_{\alpha\alpha}^{(n)}$. With an estimate of the Lagrange multiplier at hand the coordinates (Eq. (13)) can be adjusted. The

calculation is serial and moving from one bond to the next. Actually it is inherently serial and attempt to use multiple seeds (or multiple starting points to adjust individual bonds) are not guaranteed to converge to the same serial solution. A particle position is adjusted when bond α is refined, and it may be adjusted again when bond β is corrected. The diagonal is modified as the iterations proceed. Since the constraints are coupled via the off diagonal elements of $A_{\alpha\beta}^{(n)}$, the use of only diagonal elements misses something, and iterations are required. Updating the coordinates for iteration $(n+1)$, and regenerating the diagonal elements introduces a desired coupling between the bonds. This process of “bond relaxation” is known to converge quite rapidly for sparse matrices, typical of biological polymers. The number of iterations is however significant. Consider DHFR [13], which is the protein dihydrofolate reductase. When solvated in explicit water molecules in a periodic box, it is a standard benchmark system of Molecular Dynamics codes (<http://ambermd.org/amber8.bench2.html>). In DHFR we need ~50 iterations to reach relative

accuracy -- $\left| (r_{ij}^2 - r_{ij,0}^2) / r_{ij,0}^2 \right|$ of about 10^{-12} for double precision calculations. Significantly lower accuracy causes major energy drift in the tens of nanoseconds time scale and the simulation cannot be used to generate configurations in the micro-canonical ensemble.

During parallelization the total task T is divided between P processors. Some overhead O in communication, less than ideal load balancing, and dividing the task is expected. The overhead must be therefore significantly smaller than T/P for the parallelization to be effective.

The relatively large number of iterations that must be executed serially and the relatively small amount of work that needs to be done in each iteration makes the parallelism of the “bond relaxation” version of the SHAKE algorithm a very difficult task. While attempts of parallelizing “bond relaxation” were made in the past, they did not catch on within the community [14].

It turns out that satisfying the requirements for efficient parallelism is not easy with the algorithm described above. With the exception of a few attempts described below the solution was to leave the general SHAKE algorithm serial. Instead, the constraints were redefined to make trivial parallelization possible, for example, by considering only bonds that include hydrogen atoms. If independent blocks of constraints approximate the fully coupled matrix, parallelization is achieved by trivially assigning these blocks to different cores. However, reducing the constraints to bonds that include hydrogen atoms prevents the increase of the time step to more than 1 femtosecond without significant reduction in energy conservation. It also makes it difficult to simulate more tightly bound systems (more bonds per atom) than found in linear polymers or small molecules. We therefore discuss below attempts to parallelize systems with reasonably high density of constraints with a corresponding A matrix that cannot be easily broken into blocks.

III.3 Parallelism of SHAKE based on the matrix $A_{\alpha\beta}$

Successful attempts of parallelization SHAKE are based on more direct use of the matrix $A_{\alpha\beta}^{(n)}$. This is the second “philosophy” of the SHAKE algorithm that we are building on in the present manuscript. The use of the full matrix reduces significantly the number of iterations and makes the alternative algorithm more suitable for parallelism. For the DHFR system, the number of iterations needed to retain the same level of accuracy as mentioned above is ~4. On the other hand the presence of the off diagonal elements complicates the calculation, the storage, and the manipulation of the matrix, and hence the determination of the Lagrange multipliers. In practice, on a serial computer the “bond relaxation” algorithm is found to be significantly faster than SHAKE based on the full matrix (which we call here Matrix-Shake

or MSHAKE in short). Only by effective parallelization does the scale of speed tilt in favor of MSHAKE compared to bond relaxation in terms of clock time.

III.3.1 General linear solvers—The first most straightforward use of MSHAKE is by Merz et al. [15] in which off-the-shelf parallel linear solver tools were used to parallelize the process. However, it is possible to exploit some properties of the $A_{\alpha\beta}^{(n)}$ to make the calculation significantly more efficient. We first note that the matrix needs to be generated at every iteration since it depends on our current approximation for the new coordinate set, $X_i^{(n)}$ (Eq. (16)). Can we save some calculations by not computing the matrix at each MSHAKE iteration? Barth et al [2] proposed a solution in the context of a general serial algorithm. Instead of computing $A_{jk,pq}^{(n)}$ we compute a symmetric matrix, which was introduced for general constraints just below equation (7)

$$A_{jk,pq} = 2\Delta t^2 \begin{cases} \text{bond is shared } (jk=pq) & r_{jk} \cdot r_{jk} \left(\frac{1}{m_j} + \frac{1}{m_k} \right) \\ \text{atom is shared } (j=p) & r_{jk} \frac{1}{m_j} r_{jq} \\ \text{no sharing} & 0 \end{cases} \quad (21)$$

The definition in Eq. (21) is different from Eq. (14) since we use the coordinate set from the previous time step and not the coordinate set after n MSHAKE iterations. This is clearly an approximation but since the n -th iteration is different from the 0-th iteration by a term proportional to Δt^2 the error is small. There are two main advantages writing the matrix in this particular form. The first, which is probably the most important, is that the matrix is now a constant during the MSHAKE iterations of one time step. It no longer depends on the iteration number and therefore can be computed only once per time step. Fewer communications are now required since the matrix is fixed per time step. The second advantage of the above matrix is that it is symmetric and non-negative. As a symmetric matrix it requires significantly less storage, and algorithms tailored for symmetric, non-negative matrices can be used.

III.3.2 Conjugate gradient—Weinbach and Elber [3] exploit the properties of $A_{jk,pq}$ to propose a conjugate gradient (CG) algorithm to determine the Lagrange multipliers λ . CG is

an efficient approach to minimize quadratic functions of the type $F(\lambda) = \frac{1}{2} \lambda' A \lambda - \lambda' \sigma$ where A is a positive definite matrix and λ and σ are vectors. The length of the vectors and the size of the matrix are equal to the number of constraints. The matrix A can be shown to be non-negative [3] and therefore accessible to CG. In brief we search for a minimum of the function in which $dF/d\lambda = 0$, exactly our linear problem. The vector λ at the minimum is the vector of Lagrange coefficients that we seek to solve the linear problem. The advantage of CG is that history of the minimization (the value of λ , the gradient of the function, and the value of the function) is used in the calculation of a new step [16]. This additional information guarantees convergence to the minimum in L steps (L is the number of constraints). No such guarantee is available for methods that do not use the history of the optimization, such as Steepest Descent. In practice the number of iterations is much smaller than the number of constraints and for problems similar to the DHFR mentioned above it is ~ 6 . Hence, the use of an approximate matrix is not for free. The calculation is less efficient using a symmetric matrix than a calculation with the asymmetric matrix. A few more iterations are required. However, the significant gain associated with the single calculations of the matrix per time step make the additional iterations worth the cost.

Another variant of CG that allows for more efficient use is the application of pre-conditioners [17]. If we have a reasonable guess to the inverse of the matrix, we can readily apply it for additional speed up. Let $\tilde{A}^{-1}$ be an approximation to the inverse of A . We conditioned the matrix $\hat{A} = \tilde{A}^{-1/2} A \tilde{A}^{-1/2}$ to be closer to identity (for which the solution of the linear problem would be trivial). With $\hat{A}$ in hand we write $F(\lambda) = \lambda^t \tilde{A}^{1/2} \hat{A} \tilde{A}^{1/2} \lambda - \lambda^t \tilde{A}^{1/2} \tilde{A}^{-1/2} \sigma$. Defining further, $\eta = \tilde{A}^{1/2} \lambda$ and $\bar{\sigma} = \tilde{A}^{-1/2} \sigma$ we have $F(\eta) = \eta^t \hat{A} \eta - \eta^t \bar{\sigma}$. If the pre-conditioner is chosen wisely the last equation is much easier to solve. The pre-conditioner should also be simple enough to apply. A common choice is to have $\tilde{A}$ the diagonal part of A . In this case, computing the inverse is trivial and requires only L operations.

Of the three criteria mentioned above for effective parallelization of SHAKE the above algorithm satisfies easily the first and third. In [3] we consider two molecular systems: Myoglobin in vacuum and a lipid membrane. The simulations were conducted for 2000 steps on Tera, a parallel machine that no longer exists, with slow communication speed and high latency compared to modern systems. Myoglobin is clearly a worst-case scenario. It is relatively small, only 1563 atoms, and the constraint matrix cannot be broken into blocks. The equations of motions were integrated with 1fs for a time step. While the number of matrix elements that require transmission between the cores was small (the constraint matrix is highly sparse), the latency was significant and limited the efficiency of the parallelization. The load balancing was good. On one core the SHAKE calculations took 87.91s, the accumulated execution time of two processors was 93.92s, of four processor 104.48s and at 16 processors 115.01. The results suggest that the task was partitioned successfully and overhead, if exists, is a result of communication overhead. Indeed, the parallelization of the myoglobin simulation did not provide a speed up larger than a factor of 2. The results were brighter for the membrane system, where the speed up was 3.33 on four processors, 6.35 on eight, and 11.19 on 16 processors.

III.3.3 LINCS—A use of diagonal pre-conditioners is particularly useful for the system at hand since the matrix is “almost” diagonal and $\tilde{A}^{-1}$ is a reasonable starting approximation for the inverse. Further exploitation of the nature of A was made in the LINCS algorithm and its parallel version P-LINCS [4, 5]. Consider again the task at hand of determining the Lagrange multipliers given the symmetric constraint matrix and the current errors $A\lambda = \sigma$. Since A is close to diagonal it is suggested to write it as a sum of two matrices $A = D + O$ where D represents the diagonal part of A , and O the off diagonal part. Computing the inverse of the diagonal part is trivial again (like the pre-conditioner mentioned before). We have

$$\begin{aligned} (D+O)\lambda &= \sigma \\ D^{1/2}(1+D^{-1/2}OD^{-1/2})D^{1/2}\lambda &= \sigma \\ D^{1/2}\lambda &= [(1+D^{-1/2}OD^{-1/2})]^{-1}D^{-1/2}\sigma \end{aligned} \quad (22)$$

The only term that requires further consideration is $(1 + D^{-1/2}OD^{-1/2})^{-1}$. We choose the above expansion to ensure the symmetry of the matrices. The matrix O is highly sparse, however, the inverse of the matrix $(1 + D^{-1/2}OD^{-1/2})^{-1}$ is not necessarily so. Since the number of bond constraints can be substantial (typically it is in the thousands for protein molecules) we must take full advantage of the sparsity of the matrix. The following expansion suggested by Hess [4, 5] is particularly useful in the present case since typically the eigenvalues of $D^{-1/2}OD^{-1/2}$ are smaller the one. Then we can write

$$(1 + D^{-1/2} O D^{-1/2})^{-1} = 1 - D^{-1/2} O D^{-1/2} + \frac{1}{2} (D^{-1/2} O D^{-1/2})^2 - \dots \quad (23)$$

The range of influence of the off diagonal elements is increasing with the power of the expansion (or in other words, the non-zero off diagonal elements are found further from the diagonal). However, they are also getting significantly smaller and the series converges. For a fixed order of expansion it is possible to determine which of the matrix elements will be non-zero at the beginning of the calculations and to use this knowledge for accurate partition of the work load between the cores. Hence LINCS allows for a particularly efficient and parallelizable solution of the Lagrange multipliers. In the simulation of lysozyme in water [5] the accumulated time of the constraint parallelization was good. For example, one processor requires 2.4ms, the aggregated time of four processor 3.8ms, sixteen processors 5.7ms, and thirty two processors 8.8ms.

It is interesting to note that a similar efficient partitioning of loads is possible also in the CG framework. Steps in CG require the multiplication of the matrix A by a vector. Similar to the LINCS algorithm, the load of matrix-vector multiplication in CG can be divided precisely between different cores, (the constraints are divided between the processor in the CG MSHAKE approach). The matrix vector multiplication can be executed in almost perfect parallelism, since the matrix is sparse and only a few elements are shared between cores. The disadvantage of CG compared to the approach used in LINCS is that it is harder to determine where to stop (the number of iterations required to solve the linear equation is not known). The number of iterations is determined only in retrospect when the target function $F(\lambda)$ is minimal and not according to a pre-determined expansion order.

III.3.4 Further exploitation of approximate constraint matrices—We have shown that an approximate matrix formulation enables efficient parallelization of the determination of the Lagrange multipliers. The task of adjusting N coordinate and checking that L constraints are satisfied is trivially distributed between the cores. Typically, the coordinates and not the constraints are divided between the processors. The bottleneck remains the efficient implementation of sparse-matrix vector multiplication and is addressed differently in algorithms that we discussed. Concrete implementation will depend on the specific hardware we have at hand. For example, while the amount of data that requires communication is small (it includes the few shared atoms between bond constraints assigned to one core and another set of bond constraints assigned to another core), efficiency does depend on latency. Latency, which is the initiate time for a parallel call, can be costly on computer clusters and on other models of cloud and distributed computing that are not optimized for tight parallel processing.

On the other hand, a useful renewed direction is the use of shared memory machines with a significant number of processors (or cores). Fifty cores on a single CPU is a recent computer hardware announced by Intel. Shared memory parallel systems do not have significant latencies or communication times. Only load balancing remains a concern. However with static bonded structure, partitioning of bonded domains to different processors can be achieved with almost perfect load balancing solving the problem of parallel SHAKE.

We have reduced the number of calculations of the matrix A to one in a time step. Can the number of calculations be reduced further?

Imagine that we constrain both the bonds and the angles. We constrain the angles by adding a third “pseudo” bond between the extreme atoms of an angle. For example, in a water molecule we add a bond constraint between the two hydrogen atoms. The basic idea is that

angles in an empirical force field of complex molecules are modeled with harmonic potentials that restrain their fluctuations to (at most) several degrees. The angle oscillates near a single equilibrium value and in numerous modeling studies (of the Monte Carlo type) are even set rigid. The polymer dynamics of these models is restricted to torsion space. It is suggested to extend this model also to Molecular Dynamics allowing for a denser form of constraints. Eliminating bonds and angles terms will further reduce fast degrees of freedom and is expected to make it possible to further increase the time step.

In [3] it was argued that if the pseudo bonds that induced angle constraints are added to the constraint list then the symmetric matrix A could be made constant, throughout the simulation. Assume that the previous step was perfectly integrated to satisfy the constraints of the regular and pseudo bond constraints. Eq. (17) is modified to

$$A_{jk,pq} = 2\Delta t^2 \begin{cases} \text{bond is shared } (jk=pq) & r_{jk,0}^2 \left(\frac{1}{m_j} + \frac{1}{m_k} \right) \\ \text{atom is shared } (j=p) & r_{jk,0} \frac{1}{m_j} r_{jq,0} \cos(\theta_{jkq,0}) \\ \text{no sharing} & 0 \end{cases} \quad (24)$$

where $r_{ij,0}$ is the ideal bond length of the ij bonded pair. Note that we do not use a constraint of the type $\cos(\theta_{ijk}) - \cos\theta_{ijk,0} = 0$ but rather impose it by constraining the three bonds ij , jk and ik . The constraint matrix in Eq. (24) is advantageous since it is exactly constant independent of the time of integration. It can be created at the beginning of the calculation and used either in LINCS or CG frameworks. This will provide significant additional savings in that the matrix will not require recalculation at each step. A recent paper [18] claims to use the same angle constraint proposed in [3]. However, their implementation does not include a bond between the atoms at the edges of the angles. A three-atom constraint leads to a coupling to a fourth atom, an effect not considered in the above publication. Avoiding the additional bond constraint leads to an algorithm with unknown convergence properties.

It should be noted that the addition of “angle” bonds makes the problem more complex to handle. The matrix is significantly less sparse and the O matrix frequently has eigenvalues larger than 1. This makes it impossible to use the LINCS expansion but the CG approach is still appropriate.

We also comment that fixing angles increases the rigidity of the molecule and torsion transitions are likely to face larger barriers. A possible solution to this problem is the design of a new potential with reduced 1–4 interactions and torsion barriers to enable more rapid transitions. On the other hand Monte Carlo algorithms in torsion space were used in the past to sample states of biological macromolecules [19] and are an efficient mean to explore conformation space. It is expected that the efficiency of Molecular Dynamics for these cases would be comparable.

IV. Summary

We described the SHAKE algorithm, and why parallelization of it becomes necessary and timely. The usual bond relaxation procedure is not appropriate for parallelization and alternative approaches, based on the matrix formulation of SHAKE are desired. Exploiting the sparse nature of the matrix in a way amenable to easy parallelization is not trivial, since communication between the processors is required for constraints that share atoms. While the amount of data transfer is small, initiating a communication and synchronizing is costly. So far only the P-LINCS and conjugate gradient solution of the constrained matrix were illustrated to be useful.

Acknowledgments

This research was supported by NIH grant GM59796 to RE.

References

1. Ryckaert JP, Ciccotti G, Berendsen HJC. Numerical integration of cartesian equations of motion of a system with constraints - molecular dynamics of N-alkanes. *Journal of Computational Physics*. 1977; 23(3):327–341.
2. Barth E, et al. ALGORITHMS FOR CONSTRAINED MOLECULAR-DYNAMICS. *Journal of Computational Chemistry*. 1995; 16(10):1192–1209.
3. Weinbach Y, Elber R. Revisiting and parallelizing SHAKE. *Journal of Computational Physics*. 2005; 209(1):193–206.
4. Hess B, et al. LINCS: A linear constraint solver for molecular simulations. *Journal of Computational Chemistry*. 1997; 18(12):1463–1472.
5. Hess B. P-LINCS: A parallel linear constraint solver for molecular simulation. *Journal of Chemical Theory and Computation*. 2008; 4(1):116–122.
6. Bailey AG, Lowe CP. MILCH SHAKE: An Efficient Method for Constraint Dynamics Applied to Alkanes. *Journal of Computational Chemistry*. 2009; 30(15):2485–2493. [PubMed: 19373833]
7. Bailey AG, Lowe CP, Sutton AP. Efficient constraint dynamics using MILC SHAKE. *Journal of Computational Physics*. 2008; 227(20):8949–8959.
8. Gonnet P. *P-SHAKE: A quadratically convergent SHAKE in $O(n(2))$* . *Journal of Computational Physics*. 2007; 220(2):740–750.
9. Andersen HC. RATTLE - A VELOCITY VERSION OF THE SHAKE ALGORITHM FOR MOLECULAR-DYNAMICS CALCULATIONS. *Journal of Computational Physics*. 1983; 52(1):24–34.
10. Harvey MJ, Giupponi G, De Fabritiis G. ACEMD: Accelerating Biomolecular Dynamics in the Microsecond Time Scale. *Journal of Chemical Theory and Computation*. 2009; 5(6):1632–1639.
11. Stone JE, et al. Accelerating molecular modeling applications with graphics processors. *Journal of Computational Chemistry*. 2007; 28(16):2618–2640. [PubMed: 17894371]
12. Elber R, et al. Moil a program for simulations of macromolecules. *Computer Physics Communications*. 1995; 91(1–3):159–189.
13. Bowers, KJ., et al. Scientific Computing. Tampa, Florida: IEEE; 2006. Scalable ALgorithms for Molecular DYnamics Simulaions on Commodity Clusters.
14. Debolt SE, Kollman PA. AMBERCUBE MD PARALLELIZATION OF AMBERS MOLECULAR-DYNAMICS MODULE FOR DISTRIBUTED-MEMORY HYPERCUBE COMPUTERS. *Journal of Computational Chemistry*. 1993; 14(3):312–329.
15. Mertz JE, et al. VECTOR AND PARALLEL ALGORITHMS FOR THE MOLECULAR-DYNAMICS SIMULATION OF MACROMOLECULES ON SHARED-MEMORY COMPUTERS. *Journal of Computational Chemistry*. 1991; 12(10):1270–1277.
16. Fletcher, R. Practical methods of optimization. John Wiley & Sons; 2000. p. 436
17. Nocedal, J.; Wright, SJ. Numerical Optimization. In: Glynn, P.; Robinson, SM., editors. Springer Series in Operation Research. New York: Springer; 1999.
18. Eastman P V, Pander S. Constant Constraint Matrix Approximation: A Robust, Parallelizable Constraint Method for Molecular Simulations. *Journal of Chemical Theory and Computation*. 2010; 6(2):434–437. [PubMed: 20563234]
19. Noguti T, Go N. STRUCTURAL BASIS OF HIERARCHICAL MULTIPLE SUBSTATES OF A PROTEIN. 2. MONTE-CARLO SIMULATION OF NATIVE THERMAL FLUCTUATIONS AND ENERGY MINIMIZATION. *Proteins-Structure Function and Genetics*. 1989; 5(2):104–112.