Journal of Computational Physics 525 (2025) 113743

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Numerical experiments using the barycentric Lagrange treecode to compute correlated random displacements for Brownian dynamics simulations

Lei Wang [a,*], Robert Krasny [b]

[a] Department of Mathematical Sciences, University of Wisconsin-Milwaukee, Milwaukee 53211, USA
[b] Department of Mathematics, University of Michigan, Ann Arbor 48109, USA

ARTICLE INFO

Keywords:
Brownian dynamics
Hydrodynamic interactions
Correlated random displacements
RPY tensor
Matrix square root
Krylov subspace
Lanczos iteration
Barycentric Lagrange treecode

ABSTRACT

To account for hydrodynamic interactions among solvated molecules, Brownian dynamics simulations require correlated random displacements $\mathbf{g} = D^{1/2}\mathbf{z}$, where D is the $3N \times 3N$ Rotne-Prager-Yamakawa diffusion tensor for a system of N particles and $\mathbf{z}$ is a standard normal random vector. The Spectral Lanczos Decomposition Method (SLDM) computes a sequence of Krylov subspace approximations $\mathbf{g}_k \to \mathbf{g}$, but each step requires a dense matrix-vector product $D\mathbf{q}$ with a Lanczos vector $\mathbf{q}$, and the $O(N^2)$ cost of computing the product by direct summation (DS) is an obstacle for large-scale simulations. This work employs the barycentric Lagrange treecode (BLTC) to reduce the cost of the matrix-vector product to $O(N \log N)$ while introducing a controllable approximation error. Numerical experiments compare the performance of SLDM-DS and SLDM-BLTC in serial and parallel (32 core, GPU) calculations.

1. Introduction

Brownian dynamics (BD) is a coarse-grained computational technique used to study the diffusion and association of solvated molecules while accounting for solvent-mediated hydrodynamic interactions [1–3]. In this approach the molecules are represented by a set of N particles (also called beads), $\mathbf{x}_i \in \mathbb{R}^3, i = 1 : N$. The Ermak-McCammon algorithm [4] updates the configuration $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N) \in \mathbb{R}^{3N}$ by the following rule,

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \frac{\Delta t}{k_B T} D\mathbf{f} + (\nabla \cdot D)\Delta t + \mathbf{g}, \quad \langle \mathbf{g} \rangle = 0, \quad \langle \mathbf{g}\mathbf{g}^T \rangle = 2\Delta t D, \tag{1}$$

where Δt is the time step, k_B is Boltzmann's constant, T is temperature, D is a $3N \times 3N$ diffusion tensor, $\mathbf{f}$ is an external force vector, and $\mathbf{g}$ is a vector of correlated random displacements with mean zero and covariance matrix $2\Delta t D$. A common choice for D is the Rotne-Prager-Yamakawa (RPY) tensor [5,6] which is composed of 3×3 submatrices $D(\mathbf{x}_i, \mathbf{x}_j)$ for $i, j = 1 : N$ defined by

$$D(\mathbf{x}_i, \mathbf{x}_i) = \frac{k_B T}{6\pi\eta a} I_3, \tag{2a}$$

* Corresponding author.
E-mail addresses: wang256@uwm.edu (L. Wang), krasny@umich.edu (R. Krasny).

https://doi.org/10.1016/j.jcp.2025.113743
Received 1 December 2023; Received in revised form 19 December 2024; Accepted 10 January 2025

Available online 14 January 2025

$$D(\mathbf{x}_i,\mathbf{x}_j)=\frac{k_BT}{8\pi\eta x_{ij}}\left[I_3+\hat{\mathbf{x}}_{ij}\hat{\mathbf{x}}_{ij}+\frac{2a^2}{x_{ij}^2}\left(\frac{1}{3}I_3-\hat{\mathbf{x}}_{ij}\hat{\mathbf{x}}_{ij}\right)\right],\quad i\neq j,\quad x_{ij}\geq 2a, \tag{2b}$$

$$D(\mathbf{x}_i,\mathbf{x}_j)=\frac{k_BT}{6\pi\eta a}\left[\left(1-\frac{9}{32}\frac{x_{ij}}{a}\right)I_3+\frac{3}{32}\frac{x_{ij}}{a}\hat{\mathbf{x}}_{ij}\hat{\mathbf{x}}_{ij}\right],\quad i\neq j,\quad x_{ij}<2a, \tag{2c}$$

where η is the solvent viscosity, a is the particle radius, I_3 is the 3×3 unit tensor, and $\hat{\mathbf{x}}_{ij}=\mathbf{x}_{ij}/x_{ij}$ is the particle-particle unit vector with $\mathbf{x}_{ij}=\mathbf{x}_i-\mathbf{x}_j, x_{ij}=|\mathbf{x}_{ij}|$. The RPY tensor satisfies $\nabla\cdot D=0$ and is symmetric positive-definite for any particle configuration. The main cost in BD simulations is due to computing the matrix-vector product $D\mathbf{f}$ and the correlated random displacements $\mathbf{g}$; the next two subsections describe several methods for reducing the cost.

1.1. Fast methods for matrix-vector product

The matrix-vector product $D\mathbf{f}$ required in each time step of a BD simulation can be written as

$$(D\mathbf{f})_i=\sum_{j=1}^{N}D(\mathbf{x}_i,\mathbf{x}_j)\mathbf{f}_j,\quad i=1:N,\quad \mathbf{f}=(\mathbf{f}_1,\ldots,\mathbf{f}_N)^T, \tag{3}$$

where the kernel function $D(\mathbf{x}_i,\mathbf{x}_j)$ gives the pairwise interaction between a target particle $\mathbf{x}_i$ and a source particle $\mathbf{x}_j$. Since the RPY tensor is dense, direct summation of Eq. (3) requires $O(N^2)$ operations which is prohibitively expensive when N is large. This type of N-body problem occurs throughout computational physics and several methods have been developed to reduce the operation count to $O(N\log N)$ or $O(N)$. Among the earliest of these methods for gravitational and electrostatic interactions, the treecode [7] and Fast Multipole Method (FMM) [8] employed analytic kernel approximations and later on these methods were extended to particle interactions involving Stokes kernels [9]. There have also been parallel implementations of these methods for Stokes kernels [10,11] and the RPY tensor [12]. In addition, kernel-independent methods using alternative approximations were developed including the KIFMM [13], black-box FMM [14], and H-matrix methods [15,16], and several of these have been applied to matrix-vector products with the RPY tensor [17,18].

1.2. Fast methods for correlated random displacements

The vector of correlated random displacements can be obtained as

$$\mathbf{g}=B\mathbf{z}, \tag{4}$$

where $\mathbf{z}$ is a standard normal random vector and B is any matrix satisfying $BB^T=D$ (we have set $2\Delta t=1$ to simplify notation). One option for B is the lower triangular Cholesky factor of D, but the factorization requires $O(N^3)$ operations so this approach is restricted to relatively small systems [4,19,20]. Another option is the positive-definite square root matrix $D^{1/2}$, which can be computed from the spectral factorization $D=V\Lambda V^T$, where Λ is the diagonal matrix of eigenvalues and V is the orthogonal matrix of eigenvectors. In this case the correlated random displacements are expressed as

$$\mathbf{g}=D^{1/2}\mathbf{z}=V\Lambda^{1/2}V^T\mathbf{z}, \tag{5}$$

where $\Lambda^{1/2}$ is the diagonal matrix of positive square roots of the eigenvalues of D.

For large systems however, computing the spectral factorization of D is expensive and this motivated the development of methods which produce a sequence of approximations $\mathbf{g}_k\to\mathbf{g}$. In principle these methods become exact for $k=3N$, but in practice they are used for $k<<3N$. Fixman's method [21] sets $\mathbf{g}_k=p_k(D)\mathbf{z}$, where $p_k(x)$ is the kth degree Chebyshev polynomial approximation to $x^{1/2}$ on an interval containing the spectrum of D. Each step requires a matrix-vector product with D which can be accelerated by the FMM [22], but this approach also requires bounds on the spectrum of D and this is an additional cost [23]. Another option called the Spectral Lanczos Decomposition Method (SLDM) computes an approximation $\mathbf{g}_k$ in the k-dimensional Krylov subspace based on D and $\mathbf{z}$ using Lanczos iteration; see [24–26] and references therein. As in Fixman's method, each SLDM step requires a matrix-vector product with D, but bounds on the spectrum are not needed. Several studies have applied the FMM, KIFMM, and H-matrix methods to accelerate the matrix-vector product in SLDM calculations [17,12,18]. Other efficient methods for computing correlated random displacements include the Truncated Expansion Approximation (TEA) which produces an approximation to $\mathbf{g}$ under the assumption of weak hydrodynamic coupling [27,28], and a molecule-centered method that treats intermolecular hydrodynamic interactions more simply than intramolecular interactions [29].

1.3. Present work

Here we employ a recently developed fast summation method, the barycentric Lagrange treecode (BLTC), to accelerate the matrix-vector product in SLDM calculations with the RPY tensor. The BLTC uses barycentric Lagrange interpolation at Chebyshev points to compute well-separated particle-cluster interactions [30,31]. The method is kernel-independent with relatively simple implementation and low memory requirements, and it was previously applied to compute matrix-vector products with the generalized RPY tensor [32, 33]. The present work is its first application to computing correlated random displacements using the SLDM.

The remainder of the article is organized as follows. Section 2 describes the Spectral Lanczos Decomposition Method (SLDM) for computing correlated random displacements in the form $\mathbf{g} = D^{1/2}\mathbf{z}$. Section 3 describes the barycentric Lagrange treecode (BLTC) for the matrix-vector product needed in each step of the SLDM. Section 4 discusses implementation details. Section 5 presents the error and run time for the BLTC applied a single matrix-vector product. Section 6 presents serial and parallel SLDM-BLTC calculations for systems with up to $N = 1e7$ particles. The results are summarized in Section 7.

2. Spectral Lanczos decomposition method (SLDM)

The goal is to compute correlated random displacements in the form $\mathbf{g} = D^{1/2}\mathbf{z}$, where D is the RPY tensor and $\mathbf{z}$ is a standard normal vector. The vector $\mathbf{g}$ can be computed by spectral factorization as in Eq. (5), but this is expensive for large systems and the SLDM can be employed to efficiently compute an approximation as follows [25,26,17,18]. Consider the k-dimensional Krylov subspace,

$$K_k(D,\mathbf{z}) = \text{span}\{\mathbf{z}, D\mathbf{z}, \ldots, D^{k-1}\mathbf{z}\}, \quad k = 1:3N. \tag{6}$$

Lanczos iteration produces an orthonormal basis $\{\mathbf{q}_1, \ldots, \mathbf{q}_k\}$ for $K_k(D,\mathbf{z})$ and a $k \times k$ symmetric positive-definite tridiagonal matrix,

$$T_k = \begin{pmatrix} \alpha_1 & \beta_1 & \cdots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \\ \vdots & & \vdots & \\ 0 & & \beta_{k-1} & \alpha_k \end{pmatrix}, \tag{7}$$

where $T_k = Q_k^{\mathrm{T}} D Q_k$ and Q_k is the $3N \times k$ matrix whose columns are the Lanczos vectors. Note that $\mathbf{q}_1 = \mathbf{z}/||\mathbf{z}||$ and $\mathbf{z} = ||\mathbf{z}|| Q_k \mathbf{e}_1^k$, where the vector 2-norm is taken here and below and $\mathbf{e}_1^k$ is the first column of the $k \times k$ identity matrix. Then projecting $\mathbf{g}$ onto the k-dimensional Krylov subspace yields an approximation,

$$\mathbf{g}_k^* = Q_k Q_k^T \mathbf{g} = Q_k Q_k^T D^{1/2}\mathbf{z} = ||\mathbf{z}|| Q_k Q_k^T D^{1/2} Q_k \mathbf{e}_1^k. \tag{8}$$

This is still expensive because it requires $D^{1/2}$, but the cost can be reduced as follows. Consider the spectral factorization of the tridiagonal Lanczos matrix,

$$T_k = P_k \Sigma_k P_k^T, \tag{9}$$

where Σ_k is the diagonal matrix of eigenvalues and P_k is the orthogonal matrix of eigenvectors. Then the inner term in Eq (8) can be written as

$$Q_k^T D^{1/2} Q_k = Q_k^T V \Lambda^{1/2} V^T Q_k \approx Q_k^T (Q_k P_k) \Sigma_k^{1/2} (Q_k P_k)^T Q_k = P_k \Sigma_k^{1/2} P_k^T, \tag{10}$$

where $Q_k^T Q_k = I_k$ is used, and two approximations were made, (1) the eigenvalues of D are approximated by the eigenvalues of T_k, and (2) the eigenvectors of D are approximated by the columns of $Q_k P_k$. Substituting Eq. (10) into Eq. (8) yields the SLDM approximation for the correlated random displacements,

$$\mathbf{g}_k = ||\mathbf{z}|| Q_k P_k \Sigma_k^{1/2} P_k^T \mathbf{e}_1^k. \tag{11}$$

The scheme is given in Algorithm 1. Lines 3-9 comprise the Lanczos iteration with reorthogonalization in line 10 to stabilize the scheme in finite precision arithmetic [34]. Line 12 is the spectral factorization of the tridiagonal Lanczos matrix T_k which is inexpensive for $k << 3N$. Line 14 gives the stopping criterion which requires the relative increment between successive iterations to be less than the user-specified tolerance τ. The computational bottleneck is the matrix-vector product $\mathbf{w} = D\mathbf{q}_k$ in line 5 which is required in each iteration. Next we describe the BLTC used to accelerate this operation.

3. Barycentric Lagrange treecode (BLTC)

It is convenient to rewrite Eq. (3) in the following form,

$$\mathbf{u}(\mathbf{x}_i) = \sum_{j=1}^{N} D(\mathbf{x}_i, \mathbf{y}_j)\mathbf{f}_j, \quad i = 1:N, \tag{12}$$

where $\mathbf{x}_i$ is a target particle and $\mathbf{y}_j$ is a source particle. In this work the target and source particles are the same, but the treecode can accommodate cases in which they are different [35]. The cost of computing $\mathbf{u}(\mathbf{x}_i)$ in Eq. (12) by direct summation scales like $O(N^2)$ and the barycentric Lagrange treecode (BLTC) reduces this to $O(N \log N)$ using particle-cluster approximations. The following subsections describe the treecode structure, barycentric Lagrange interpolation, particle-cluster approximation, and finally the BLTC algorithm.

Algorithm 1 Spectral Lanczos Decomposition Method (SLDM).

input: particles $\{\mathbf{r}_i, i = 1:N\}$, standard normal vector $\mathbf{z}$, tolerance τ
output: approximate vector of correlated random displacements $\mathbf{g}_k \approx D^{1/2}\mathbf{z}$
initialize $\mathbf{q}_0 = 0, \mathbf{q}_1 = \mathbf{z}/||\mathbf{z}||, \beta_0 = 0$
for $k = 1, 2, 3, \ldots$
  $\mathbf{w} = D\mathbf{q}_k$
  $\alpha_k = \mathbf{w}^T\mathbf{q}_k$
  $\mathbf{w} = \mathbf{w} - \alpha_k\mathbf{q}_k - \beta_{k-1}\mathbf{q}_{k-1}$
  $\beta_k = ||\mathbf{w}||$
  $\mathbf{q}_{k+1} = \mathbf{w}/\beta_k$
  reorthogonalize $\mathbf{q}_{k+1}$ against $\mathbf{q}_1, \ldots, \mathbf{q}_k$ and set $Q_{k+1} = [\mathbf{q}_1 \cdots \mathbf{q}_{k+1}]$
  if $k \geq 2$, then
    compute spectral factorization $T_k = P_k\Sigma_k P_k^T$ using LAPACK routine `dsteqr`
    $\mathbf{g}_k = ||\mathbf{z}||Q_k P_k \Sigma_k^{1/2} P_k^T \mathbf{e}_1^k$
    if $||\mathbf{g}_k - \mathbf{g}_{k-1}||/||\mathbf{g}_k|| < \tau$, then exit; else continue iteration, end if
  end if
end for

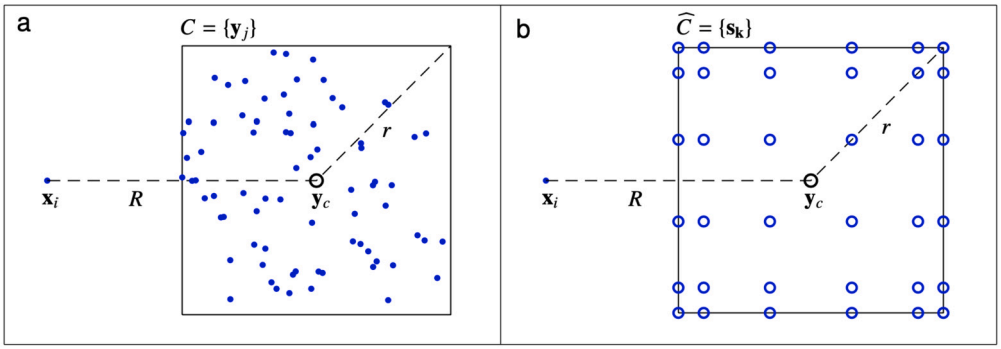

Fig. 1. Particle-cluster interaction, (a) target particle $\mathbf{x}_i$ interacts with source cluster $C = \{\mathbf{y}_j\}$, (b) target particle $\mathbf{x}_i$ interacts with proxy source cluster $\widehat{C} = \{\mathbf{s}_\mathbf{k}\}$ consisting of Chebyshev interpolation points, cluster center $\mathbf{y}_c$, cluster radius r, particle-cluster distance R.

3.1. Treecode structure

The BLTC follows the Barnes-Hut treecode structure [7], where however barycentric Lagrange interpolation is utilized in place of a monopole approximation [31]. The loop over source particles $\mathbf{y}_j$ in Eq. (12) is replaced by a loop over source clusters C,

$$\mathbf{u}(\mathbf{x}_i) = \sum_C \mathbf{u}(\mathbf{x}_i, C), \quad \mathbf{u}(\mathbf{x}_i, C) = \sum_{\mathbf{y}_j \in C} D(\mathbf{x}_i, \mathbf{y}_j)\mathbf{f}_j, \quad i = 1:N, \tag{13}$$

where $\mathbf{u}(\mathbf{x}_i, C)$ is the interaction between a target particle $\mathbf{x}_i$ and a cluster of source particles $C = \{\mathbf{y}_j\}$. In this work the clusters are 3D cubes and the set of all clusters has a tree structure as explained below. Fig. 1a depicts a particle-cluster interaction, where $\mathbf{y}_c$ is the cluster center, r is the cluster radius, and R is the particle-cluster distance. The particle-cluster interaction $\mathbf{u}(\mathbf{x}_i, C)$ can be computed directly as in Eq. (13), but if $\mathbf{x}_i$ and C are well-separated, then the interaction can be approximated using barycentric Lagrange interpolation as described below. A target particle $\mathbf{x}_i$ and source cluster C are considered to be well-separated if the multipole acceptance criterion (MAC) is satisfied,

$$r/R < \theta, \tag{14}$$

where θ is a user-specified parameter; in this case $\mathbf{x}_i$ interacts with the proxy source cluster $\widehat{C} = \{\mathbf{s}_\mathbf{k}\}$ given by a tensor product grid of Chebyshev points as depicted in Fig. 1b. Next we review barycentric Lagrange interpolation [30].

3.2. Barycentric Lagrange interpolation

Consider a function $f(t)$ on the interval $[-1, 1]$ and Chebyshev points $s_k = \cos(k\pi/n), k = 0:n$. The barycentric Lagrange form of the interpolating polynomial is

$$p(t) = \sum_{k=0}^{n} L_k(t) f_k, \quad L_k(t) = \frac{\dfrac{w_k}{t - s_k}}{\displaystyle\sum_{\ell=0}^{n} \frac{w_\ell}{t - s_\ell}}, \quad w_k = (-1)^k \delta_k, \quad \delta_k = \begin{cases} 1/2, & k = 0 \text{ or } k = n, \\ 1, & k = 1:n-1, \end{cases} \tag{15}$$

Algorithm 2 Barycentric Lagrange treecode (BLTC).

```
1: program main
2:    input: particle positions x_i, forces f_i, i = 1 : N
3:    input: treecode parameters, MAC θ, degree n, maximum leaf size N_0
4:    output: matrix-vector product u(x_i), i = 1 : N in Eq. (13)
5:    build octtree, compute proxy forces f̂_k(C) for each cluster C
6:    for i = 1 : N; pc-interaction (x_i, root); end for
7: end program main
8: subroutine pc-interaction (x_i, C)
9:    if MAC is satisfied, compute particle-cluster approximation in Eq. (17)
10:   else if C is a leaf, compute particle-cluster interaction directly by Eq. (13)
11:   else for each child C' of C; pc-interaction (x_i, C'); end for
12:   end if
13: end subroutine pc-interaction
```

where the Lagrange polynomials satisfy $L_k(s_\ell) = \delta_{k\ell}$, and the barycentric weights w_k take the indicated form in the case of Chebyshev interpolation points [36]. Polynomial interpolation at the Chebyshev points is spectrally accurate [37], and this form of the interpolating polynomial is numerically stable and efficient [30,38]. To interpolate on a general interval $[a, b]$, the Chebyshev points are linearly mapped to $[a, b]$, but the weights do not change. Barycentric Lagrange interpolation can be extended to 3D cubes by a tensor product in the Cartesian coordinates.

3.3. Particle-cluster approximation

Now consider a target particle $\mathbf{x}_i$ and a well-separated cluster C. Let $\mathbf{s_k} = (s_{k_1}, s_{k_2}, s_{k_3})$ denote the tensor product grid of Chebyshev points adapted to the cluster as shown schematically in Fig. 1b; the Chebyshev points play the role of proxy particles for the source particles $\mathbf{y}_j$ in C. In this case the RPY kernel can be approximated by barycentric Lagrange interpolation with respect to the source particle $\mathbf{y}_j = (y_{j1}, y_{j2}, y_{j3})$ keeping the target $\mathbf{x}_i$ fixed,

$$D(\mathbf{x}_i, \mathbf{y}_j) \approx \sum_{k_1=0}^{n} \sum_{k_2=0}^{n} \sum_{k_3=0}^{n} D(\mathbf{x}_i, \mathbf{s_k}) L_{k_1}(y_{j1}) L_{k_2}(y_{j2}) L_{k_3}(y_{j3}), \quad \mathbf{y}_j \in C. \tag{16}$$

Then substituting Eq. (16) into Eq. (13) and switching the order of summation yields the particle-cluster approximation,

$$\mathbf{u}(\mathbf{x}_i, C) \approx \sum_{k_1=0}^{n} \sum_{k_2=0}^{n} \sum_{k_3=0}^{n} D(\mathbf{x}_i, \mathbf{s_k}) \widehat{\mathbf{f}}_{\mathbf{k}}(C), \quad \widehat{\mathbf{f}}_{\mathbf{k}}(C) = \sum_{\mathbf{y}_j \in C} L_{k_1}(y_{j1}) L_{k_2}(y_{j2}) L_{k_2}(y_{j2}) \mathbf{f}_j, \tag{17}$$

where $\widehat{\mathbf{f}}_{\mathbf{k}}(C)$ are proxy forces associated with the proxy particles $\mathbf{s_k}$ in cluster C. The proxy forces $\widehat{\mathbf{f}}_{\mathbf{k}}(C)$ are independent of the target particle $\mathbf{x}_i$ and are computed when the tree is built. Given the proxy forces, the operation count for the particle-cluster approximation $\mathbf{u}(\mathbf{x}_i, C)$ in Eq. (17) is $O(n^3)$ independent of the number of source particles in C.

3.4. BLTC algorithm

The BLTC is described in Algorithm 2. Lines 2-3 input the particle data and treecode parameters comprising the MAC θ, degree n, and maximum leaf size N_0. Line 5 builds an octtree of particle clusters. The root cluster is a cube containing all the source particles. The root is bisected along the Cartesian axes and the eight children become subclusters of the root. The child clusters are similarly bisected, and the process continues until a cluster contains fewer than N_0 particles. The proxy forces $\widehat{\mathbf{f}}_{\mathbf{k}}(C)$ are computed along with the clusters. Line 6 is a loop over the target particles. Starting from the root cluster, the recursive subroutine **pc-interaction** computes the particle-cluster interaction $\mathbf{u}(\mathbf{x}_i, C)$. In line 9 if the MAC is satisfied, the interaction is approximated by barycentric Lagrange interpolation as in Eq. (17). Otherwise in lines 10-11 the code checks the child clusters, or if the cluster is a leaf (no children), then the interaction is computed directly by Eq. (13). The operation count is $O(N \log N)$, where the factor N arises from the loop over target particles and the factor $\log N$ reflects the number of levels in the tree. The BLTC is kernel-independent in the same sense as the KIFMM [13], black-box FMM [14], and H-matrix methods [15,16] in that it requires only kernel evaluations for arbitrary target-source pairs rather than analytic kernel expansions.

4. Implementation details

In these tests the solvent viscosity is $\eta = 1$. The particles have radius $a = 0.1$ and are uniformly randomly distributed in the cube $[0, L]^3$ with particle volume fraction defined by

$$\mathrm{PVF} = \frac{4\pi a^3 N}{3L^3}. \tag{18}$$

Under typical biomolecular conditions the particle volume fraction lies in the range $0.1 \le \mathrm{PVF} \le 0.4$ [29,39]; most of the present calculations use PVF = 0.12, but some results for PVF = 0.004, 0.04, 0.4 are also presented as noted in the text. The BLTC has three

Table 1
RPY matrix-vector product $D\mathbf{f}$, system size $N = 1e6$, particle volume fraction PVF = 0.12, BLTC MAC θ, degree n, treecode approximation error in Eq. (19), run time (s), direct sum run time = 8409 s, calculations on one CPU core.

n	$\theta = 0.9$		$\theta = 0.8$		$\theta = 0.7$		$\theta = 0.6$	
	error	run time (s)	error	run time (s)	error	run time (s)	error	run time (s)
2	4.04e-2	25	2.64e-2	32	1.64e-2	45	9.62e-3	67
3	1.62e-2	40	8.80e-3	52	4.59e-3	72	2.15e-3	106
4	6.40e-3	57	2.81e-3	74	1.19e-3	102	4.50e-4	150
5	2.90e-3	91	1.05e-3	119	3.51e-4	162	1.05e-4	240
6	1.33e-3	124	3.88e-4	162	1.03e-4	222	2.44e-5	324

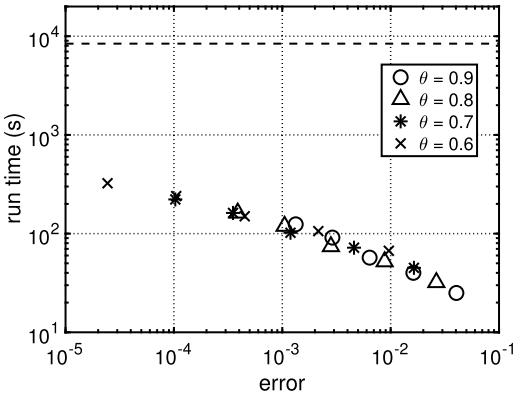

Fig. 2. RPY matrix-vector product $D\mathbf{f}$, system size $N = 1e6$, particle volume fraction PVF = 0.12, BLTC MAC θ, degree n, treecode approximation error in Eq. (19), run time (s), direct sum run time = 8409 s, calculations on one CPU core.

user-specified parameters, the multipole acceptance criterion (MAC) θ, polynomial degree n, and maximum number of particles in a leaf which is set to $N_0 = 1000$. Since the direct sum for the largest system size $N = 1e7$ is prohibitively expensive, in that case the error is computed at a subset of 2000 particles and the run time and peak memory usage are extrapolated from smaller systems. The code was written in double precision C++ and is available for download [40].

Serial and parallel CPU calculations were performed at the University of Wisconsin-Milwaukee Mortimer Faculty Research Cluster, where each node is a Dell PowerEdge R440 server with two 16-core Intel(R) Xeon(R) Gold E5-5218 processors at 2.30 GHz and 128GB RAM. The code was compiled using the Intel icpc compiler with -O2 optimization. Serial runs used a single core and parallel runs used 32 cores on a single node with OpenMP. Parallel GPU calculations were performed at the University of Michigan Great Lakes Cluster using an NVIDIA Tesla V100 GPU for the matrix-vector product (line 5 in Algorithm 1) and an Intel Xeon Gold 6148 processor at 2.4 GHz and 180GB RAM for other SLDM operations including vector inner products, reorthogonalization, and spectral factorization (lines 6-14 in Algorithm 1). The code was compiled using the NVIDIA pgc++ compiler with -fast optimization and OpenACC.

5. Numerical results: BLTC for a single matrix-vector product $D\mathbf{f}$

This section reports the BLTC performance for a single matrix-vector product $D\mathbf{f}$, where D is the RPY diffusion matrix and the forces $\mathbf{f}_i$ are sampled from a standard normal distribution with mean zero and unit variance. The particle volume fraction is PVF = 0.12. The treecode approximation error is defined by

$$\text{error} = \frac{||(D\mathbf{f})^{\text{tc}} - (D\mathbf{f})^{\text{ds}}||}{||(D\mathbf{f})^{\text{ds}}||}, \tag{19}$$

where the superscript indicates whether the matrix-vector product is computed by the treecode (tc) or direct summation (ds).

Table 1 records the error and run time (s) for several values of the MAC θ and polynomial degree n with system size $N = 1e6$. The data is also plotted in Fig. 2 to show the scaling more clearly. The BLTC becomes more accurate as the MAC θ decreases and the degree n increases, but in that case the run time also increases. For future reference we define two parameter sets, BLTC-hi with $(\theta, n) = (0.7, 6)$ yielding about 4-digit accuracy with run time 222 s, and BLTC-lo with $(\theta, n) = (0.9, 3)$ yielding about 2-digit accuracy with run time 40 s. Note that the direct sum run time in this case is 8409 s, so the treecode is much faster.

Table 2 compares the matrix-vector product using the BLTC and direct summation (DS) for system size $N = 1e5$, 1e6, 1e7. The BLTC error increases only slightly with system size. For the largest system size $N = 1e7$, the DS run time (s) and peak memory (MB) were obtained by extrapolation. The DS run time scales like $O(N^2)$ and the BLTC run time scales approximately like $O(N \log N)$. The DS memory scales like $O(N)$ corresponding to the particle data, while the BLTC memory also includes the tree data structure and $O(n^3)$ Chebyshev interpolation points in each cluster. Hence the BLTC-hi memory is larger than the BLTC-lo memory, but even in the worst case the BLTC-hi memory is only slightly more than twice the DS memory.

Table 2
RPY matrix-vector product $D\mathbf{f}$, system size $N = 1e5, 1e6, 1e7$, particle volume fraction PVF = 0.12, direct summation (DS), BLTC-hi $(\theta, n) = (0.7, 6)$, BLTC-lo $(\theta, n) = (0.9, 3)$, treecode approximation error in Eq. (19), run time (s), peak memory (MB), calculations on one CPU core.

		error	run time (s)	peak memory (MB)
$N = 1e5$	DS	–	82	8.8
	BLTC-hi	6.79e-5	13	20
	BLTC-lo	1.11e-2	2.6	11
$N = 1e6$	DS	–	8409	88
	BLTC-hi	1.03e-4	222	169
	BLTC-lo	1.62e-2	40	103
$N = 1e7$	DS	–	850772	880
	BLTC-hi	1.12e-4	3218	1479
	BLTC-lo	1.62e-2	544	996

Table 3
RPY matrix-vector product $D\mathbf{f}$, system size $N = 1e6$, particle volume fraction PVF = 0.12, direct sum (DS), BLTC-hi $(\theta = 0.7, n = 6)$, BLTC-lo $(\theta = 0.9, n = 3)$, 1 core, 32 cores, GPU, run time (s), parallel efficiency PE (%), speedup (DS run time divided by BLTC run time).

	DS		BLTC-hi			BLTC-lo		
	run time (s)	PE	run time (s)	PE	speedup	run time (s)	PE	speedup
1 core	8409	100%	222.0	100%	37×	40	100%	210×
32 cores	363	72%	13.6	51%	27×	4.5	28%	81×
GPU	42	–	1.4	–	30×	0.23	–	183×

Table 3 presents serial and parallel results for the matrix-vector product with system size $N = 1e6$ computed using DS, BLTC-hi and BLTC-lo including the run time (s), parallel efficiency PE (%), and speedup (DS run time divided by BLTC run time). With 32 cores, DS has 72% parallel efficiency, while the parallel efficiency is 51% for BLTC-hi and 28% for BLTC-lo; even so, the BLTC calculations are faster; the DS run time is 363 s, while the BLTC-hi run time is 13.6 s (27× speedup) and the BLTC-lo run time is 4.5 s (81× speedup). The GPU calculations are faster yet; the DS run time is 42 s, while the BLTC-hi run time is 1.4 s (30× speedup) and the BLTC-lo run time is 0.23 s (183× speedup).

6. Numerical results: SLDM-BLTC for correlated random displacements $D^{1/2}\mathbf{z}$

This section reports the performance of SLDM-BLTC for computing the correlated random displacements $D^{1/2}\mathbf{z}$. First we investigate the scheme's accuracy and convergence properties. Recall that the method provides a sequence of approximations $\mathbf{g}_k$ to the exact vector $\mathbf{g} = D^{1/2}\mathbf{z}$. Aside from computer roundoff error which was addressed by reorthogonalization, there are two additional sources of error in these calculations, (1) the SLDM iteration error at step k assuming the matrix-vector product is computed by direct summation, and (2) the treecode approximation error that occurs when the matrix-vector product is computed by the BLTC. The following quantities are used to measure the accuracy of the results [23,26],

$$I_k = \frac{||\mathbf{g}_k - \mathbf{g}_{k-1}||}{||\mathbf{g}_{k-1}||}, \quad E_k = \frac{||\mathbf{g}_k - \mathbf{g}_{\text{ref}}^{\text{ds}}||}{||\mathbf{g}_{\text{ref}}^{\text{ds}}||}, \quad E_k^f = \frac{|\mathbf{g}_k^T \mathbf{g}_k - \mathbf{z}^T D\mathbf{z}|}{\mathbf{z}^T D\mathbf{z}}. \tag{20}$$

In these expressions $\mathbf{g}_k$ is the SLDM vector at step k, and further below a superscript ds or tc is added to indicate whether the matrix-vector product is computed by direct summation or the BLTC. The first quantity, I_k, measures the increment between step k and step $k-1$; this is used in the stopping criterion. The second quantity, E_k, measures the error in $\mathbf{g}_k$, where the reference $\mathbf{g}_{\text{ref}}^{\text{ds}}$ was computed using direct summation for the matrix-vector product and iterating until the increment saturates close to machine precision; the error is generally not available because computing the reference is expensive. The third quantity, E_k^f, which we call the inner product error, satisfies $E_k^f = 0$ for all $k \geq 1$ in exact arithmetic [26].

6.1. Accuracy and convergence properties of SLDM-BLTC

We consider a test case with system size $N = 1e5$ and particle volume fraction PVF = 0.12. Fig. 3 shows the error E_k (solid lines) and increment I_k (dashed lines) versus step k, where the matrix-vector product is computed by direct sum (DS), BLTC-hi and BLTC-lo. Results are shown for (a) $k = 1:750$, (b) $k = 1:125$.

First consider the increment I_k. The BLTC-hi increment matches the DS increment to within plotting accuracy; they both decrease rapidly in the first few steps, then transition to exponential decay, and then saturate close to machine precision. Convergence of the

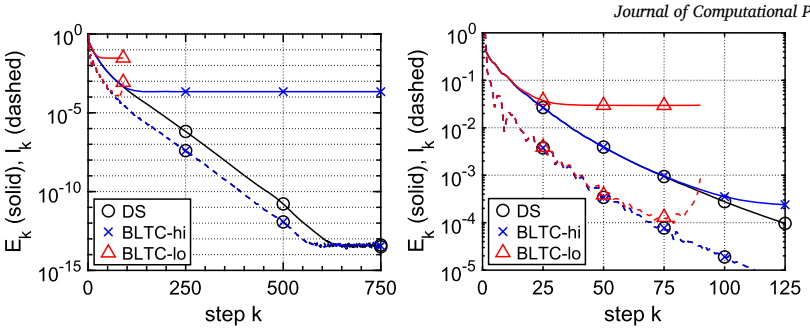

Fig. 3. SLDM for $D^{1/2}\mathbf{z}$, system size $N = 1\text{e}5$, particle volume fraction PVF $= 0.12$, error E_k (solid lines) and increment I_k (dashed lines) versus step k, (a) $k = 1:750$, (b) $k = 1:125$, matrix-vector product by direct sum (DS), BLTC-hi ($\theta = 0.7, n = 6$), BLTC-lo ($\theta = 0.9, n = 3$), BLTC-hi increment matches DS increment to within plotting accuracy.

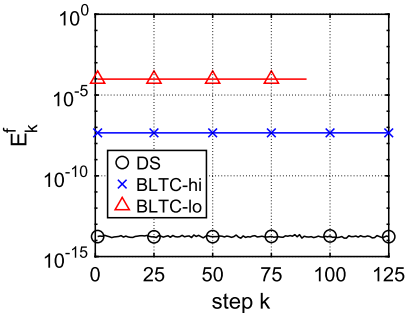

Fig. 4. SLDM for $D^{1/2}\mathbf{z}$, inner product error E_k^f versus step k, system size $N = 1\text{e}5$, particle volume fraction PVF $= 0.12$, matrix-vector product by direct sum (DS), BLTC-hi ($\theta = 0.7, n = 6$), BLTC-lo ($\theta = 0.9, n = 3$).

increments implies that the $\mathbf{g}_k^{\text{ds}}$ and $\mathbf{g}_k^{\text{tc}}$ sequences converge to well-defined limits, but as we shall see, the limit obtained using direct summation is different than the limit obtained using BLTC-hi. The BLTC-lo increment matches the DS and BLTC-hi increments until reaching level 1e-4; afterwards it increases sharply and the calculation breaks down at step $k = 92$ when the smallest eigenvalue of the Lanczos matrix T_k becomes negative; this will be discussed below.

Next consider the error E_k where the reference, $\mathbf{g}_{\text{ref}}^{\text{ds}}$, is the limit of the SLDM-DS calculation. The results show that the DS error decreases exponentially until it saturates close to machine precision, but essentially throughout the entire iteration the DS error is approximately ten times larger than the DS increment; a similar result was obtained for system size $N = 1\text{e}3$ by Ando et al. [26]. The BLTC-hi error matches the DS error until around step $k = 75$ after which it saturates at level 4e-4; this means that the $\mathbf{g}_k^{\text{tc}}$ vectors computed by BLTC-hi converge to a well-defined limit, but the limit is different than the reference vector $\mathbf{g}_{\text{ref}}^{\text{ds}}$ computed by direct summation. The BLTC-lo error matches the DS and BLTC-hi errors until around step $k = 20$, after which it saturates at level 5e-2 until breakdown occurs at step $k = 92$.

For practical reasons the SLDM stopping criterion is based on the increment, $I_k < \tau$, but when the criterion is satisfied, the error E_k is generally larger. For example in Fig. 3 if $\tau = 1\text{e}{-3}$, the stopping criterion is satisfied at step $k = 40$ for DS, BLTC-hi and BLTC-lo calculations, but the DS and BLTC-hi errors are 9 times larger than the increment, while the BLTC-lo error is 32 times larger. Hence the increment can give a false sense of confidence in the accuracy of the correlated random displacements. It should be noted that the errors E_k in the DS, BLTC-hi, and BLTC-lo calculations initially overlap for more than one order of magnitude down to roughly $E_k \approx 1\text{e}{-1}$, and hence if only low accuracy in the SLDM iterations is needed, then the BLTC-lo can be used.

Alternatively, the inner product error E_k^f has been used to monitor convergence of the Chebyshev polynomial approximation method [23]. Fig. 4 shows that in the present SLDM calculations, E_k^f is independent of the step k; the DS inner product error is close to machine precision (consistent with the result that $E_k^f = 0$ for all k in exact arithmetic [26]), while the BLTC-hi inner product error is 5e-8 and the BLTC-lo inner product error is 1e-4. Hence in these calculations the inner product error E_k^f reflects the treecode approximation error rather than the SLDM iteration error and it cannot be used as the stopping criterion.

6.2. *Eigenvalues of Lanczos matrix T_k*

Recall that step k in the SLDM requires computing the square root of the Lanczos matrix T_k, where in principle the eigenvalues of T_k are positive real numbers. Fig. 5 plots the maximum, minimum and median eigenvalues of T_k versus step k when the matrix-vector product is computed by DS, BLTC-hi and BLTC-lo. Note that the BLTC-hi eigenvalues fall on top of the DS eigenvalues. The maximum eigenvalue is almost independent of step k, while the median and minimum eigenvalues decrease slightly as the iteration proceeds. The BLTC-lo maximum and median eigenvalues match their DS and BLTC-hi counterparts until the BLTC-lo calculation breaks down

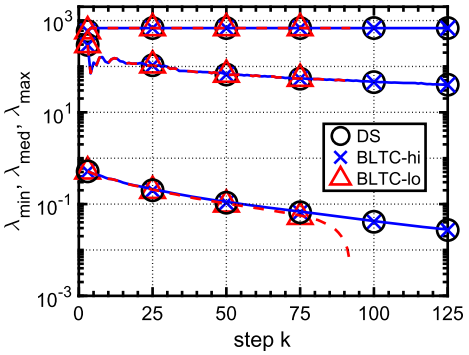

Fig. 5. SLDM for $D^{1/2}\mathbf{z}$, system size $N = 1e5$, particle volume fraction PVF $= 0.12$, eigenvalues of Lanczos matrix T_k versus step k, maximum (λ_{max}), minimum (λ_{min}), median (λ_{med}), matrix-vector product by DS (black solid line), BLTC-hi (blue solid line), BLTC-lo (red dashed line). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

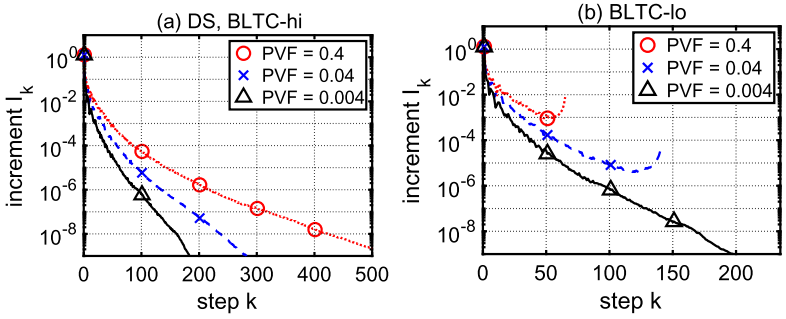

Fig. 6. SLDM for $D^{1/2}\mathbf{z}$, increment I_k versus step k, system size $N = 1e5$, particle volume fraction PVF $= 0.004, 0.04, 0.4$, matrix-vector product by (a) DS, BLTC-hi (BLTC-hi results fall on top of DS results), (b) BLTC-lo.

at step $k = 92$. The BLTC-lo minimum eigenvalue matches the others until step $k = 75$, after which it decreases sharply until step $k = 92$ when it becomes negative; at this point the calculation of the square root matrix $T_k^{1/2}$ fails and the SLDM-BLTC-lo calculation breaks down. We tried to correct this by setting the negative eigenvalues to zero, but more and more eigenvalues became negative and were to set zero, and the results became highly inaccurate.

6.3. Effect of particle volume fraction

Next we consider the effect of the particle volume fraction for $N = 1e5$ randomly distributed particles in a cube, where the cube size is varied to obtain small PVF $= 0.004$, medium PVF $= 0.04$, and large PVF $= 0.4$. Fig. 6 shows the increment I_k versus step k, where the matrix-vector product was computed by (a) DS, BLTC-hi, and (b) BLTC-lo. In frame (a) the BLTC-hi results fall on top of the DS results and the increments decrease faster for smaller PVF. In frame (b) note that the horizontal axis has been expanded. In fact the BLTC-lo increment matches the DS and BLTC-hi increments for small PVF $= 0.004$. For medium PVF $= 0.04$, the BLTC-lo increment falls below 1e-5 after which it increases sharply and breakdown occurs at step $k = 142$ as discussed above. For large PVF $= 0.4$, the BLTC-lo increment reaches level 1e-3 and breakdown occurs at step $k = 66$.

6.4. Run time versus error

Fig. 7 shows the SLDM run time (s) versus error E_k for $N = 1e5$ randomly distributed particles in a cube with PVF$= 0.12$. The matrix-vector product was computed by direct sum (DS, red symbols) and by the treecode (BLTC, blue symbols). Four combinations of SLDM tolerance and BLTC MAC were computed, $(\tau, \theta) =$ (1e-2, 0.9), (1e-3, 0.8), (1e-4, 0.7), (1e-5, 0.6), as shown along the top of the plot, and for each (τ, θ) combination, the BLTC degree n is shown in the legend and increases from right to left on each connected blue line. For each value of the tolerance, a vertical red dashed line is drawn through the DS error. The calculations were done on one CPU core.

First consider the direct sum results. Reducing the SLDM tolerance τ also reduces the DS error, but the DS run time increases because more iterations are needed to satisfy the stopping criterion. For a given tolerance τ, the DS error is about ten times larger than τ, and this is consistent with the error and increment results in Fig. 3. Next consider the BLTC results. For each combination of tolerance τ and MAC θ, the BLTC error converges to the DS error as the degree n increases. The BLTC is faster than DS; for high accuracy with $\tau =$ 1e-5, the BLTC with MAC $\theta = 0.6$ and degree $n = 6$ is 4 times faster than DS, and for low accuracy with $\tau =$ 1e-2, the BLTC with MAC $\theta = 0.9$ and degree $n = 3$ is 30 times faster than DS. Hence in SLDM calculations with a given tolerance τ,

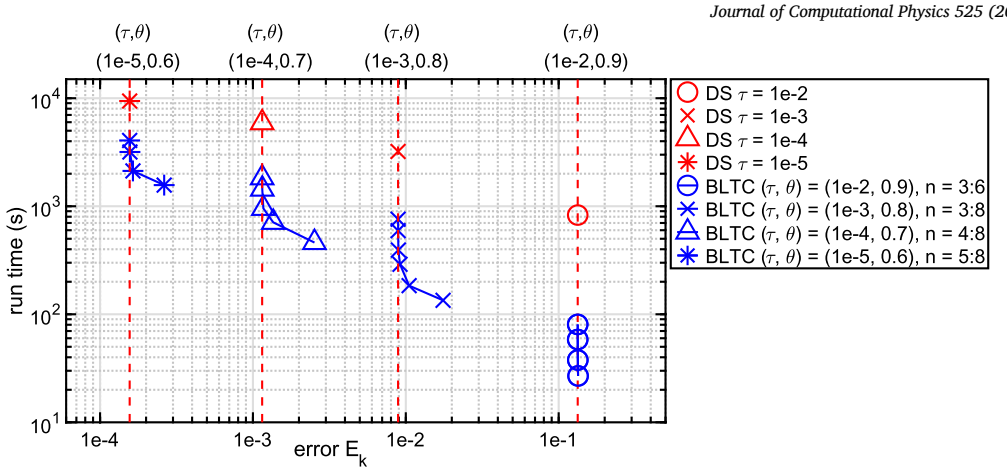

Fig. 7. SLDM for $D^{1/2}\mathbf{z}$, run time (s) versus error E_k, system size $N = 1e5$, particle volume fraction PVF $= 0.12$, matrix-vector product by direct sum (DS, red symbols) and treecode (BLTC, blue symbols), SLDM tolerance and BLTC MAC $(\tau, \theta) =$ (1e-2, 0.9), (1e-3, 0.8), (1e-4, 0.7), (1e-5, 0.6), vertical red dashed lines are drawn through DS error for each tolerance, BLTC degree n in legend increases from right to left on each connected blue line for each (τ, θ) combination, calculations on one CPU core.

Table 4
SLDM for $D^{1/2}\mathbf{z}$, system size $N =$ 1e5, 1e6, 1e7, particle volume fraction PVF $= 0.12$, matrix-vector product by direct sum (DS) and treecode (BLTC), calculations with tolerance $\tau =$ 1e-4 are denoted DS-hi, BLTC-hi ($\theta = 0.7$, $n = 6$) and those with tolerance $\tau =$ 1e-2 are denoted DS-lo, BLTC-lo ($\theta = 0.9$, $n = 3$), error E_k, iteration count (iter), run time (s) on 1 core, 32 cores, speedup (1 core run time/32 core run time), GPU run time (s), speedup (1 core run time/GPU run time), na indicates run time is more than 28 hours.

$N =$ 1e5	τ	E_k	iter	1 core	32 cores		GPU	
				run time (s)	run time (s)	speedup	run time (s)	speedup
DS-hi	1e-4	1.149e-3	72	5940	217	27.4×	18.6	319.4×
BLTC-hi	1e-4	1.170e-3	72	954	68	14.0×	36.6	26.1×
DS-lo	1e-2	1.332e-1	10	825	36	22.9×	2.7	305.6×
BLTC-lo	1e-2	1.339e-1	10	26	2.7	9.6×	1.6	16.3×
$N =$ 1e6								
DS-hi	1e-4	na	111	na	40340	na	4862	na
BLTC-hi	1e-4	na	111	24693	1574	15.7×	714	34.6×
DS-lo	1e-2	na	12	100949	4355	23.2×	517	195.3×
BLTC-lo	1e-2	na	12	478	52	9.2×	19.8	24.1×
$N =$ 1e7								
DS-hi	1e-4	na	na	na	na	na	na	na
BLTC-hi	1e-4	na	170	na	33519	na	15474	na
DS-lo	1e-2	na	na	na	na	na	na	na
BLTC-lo	1e-2	na	12	6537	658	9.9×	212	30.8×

computing the matrix-vector product using the BLTC yields a significant speedup while maintaining the same accuracy as direct summation.

6.5. Parallel performance

Table 4 records the SLDM parallel performance for system size $N =$ 1e5, 1e6, 1e7 with particle volume fraction PVF $= 0.12$. The matrix-vector product was performed using direct summation (DS) and the treecode (BLTC). Calculations with tolerance $\tau =$ 1e-4 are denoted DS-hi and BLTC-hi ($\theta = 0.7$, $n = 6$), and those with tolerance $\tau =$ 1e-2 are denoted DS-lo and BLTC-lo ($\theta = 0.9$, $n = 3$). Results are shown for 1 core, 32 cores, and 1 GPU, where in the latter case the matrix-vector product (line 5 in Algorithm 1) was done on the GPU and the other SLDM operations including vector inner products, reorthogonalization, and spectral factorization (lines 6-14 in Algorithm 1) were done on 1 CPU core. The Table shows the error E_k, iteration count (iter), run time (s) on 1 core and 32 cores), speedup (1 core run time/32 core run time), GPU run time (s), speedup (1 core run time/GPU run time). Complete results were obtained for the smaller system size $N =$ 1e5, and only partial results are available for the larger systems due the cost of the calculations; cases in which the run time is greater than 28 hours are labeled na (not available).

For system size $N =$ 1e5 and given tolerance τ, the DS and BLTC have almost the same error E_k which is roughly ten times larger than τ as seen before. For system size $N =$ 1e6, 1e7, the error is not available due to the high cost of the reference solution, but Table 2 showed that the BLTC error for a single matrix-vector product is nearly independent of N and we expect this also holds

for SLDM-BLTC. For cases in which results are available, the iteration count increases as the system size increases, but for each tolerance τ, the SLDM-DS and SLDM-BLTC have the same iteration count.

Focusing on system size $N = 1e5$ where complete results are available, DS-hi achieves the highest speedup, followed by DS-lo, then BLTC-hi and then BLTC-lo; this ordering prevails on 32 cores and the GPU, but the speedup is higher on the GPU than on 32 cores. In terms of actual run time, for high accuracy, BLTC-hi is faster than DS-hi on 32 cores, but slower than DS-hi on the GPU; this is due to several factors, (1) the direct sum parallelizes very well on the GPU, (2) the system size $N = 1e5$ is still small enough that the $O(N^2)$ scaling is not yet fully felt, and (3) the BLTC-hi is penalized by having operation count $O(n^3)$ with relatively high interpolation degree $n = 6$. The situation is different for low accuracy, where BLTC-lo is faster than DS-lo on both 32 cores and the GPU.

Where available, results in Table 4 for system size $N = 1e6, 1e7$ show the advantage of BLTC compared to DS and the advantage of the GPU compared to 32 cores for high and low accuracy. Hence combining algorithmic acceleration (due to the BLTC) with hardware acceleration (due to the GPU) provides a significant reduction in run time without sacrificing accuracy.

A final comment concerns the GPU calculation. Table 3 showed that for system size $N = 1e6$, a single matrix-vector product on the GPU has run time 1.4 s for BLTC-hi and 0.23 s for BLTC-lo. This can be used to estimate how much of the GPU run time reported in Table 4 is spent on the CPU versus the GPU. For instance with system size $N = 1e6$, the BLTC-hi calculation took 714 s and 111 iterations; assuming 1.4 s for each matrix-vector product, this implies 155.4 s for the GPU and 558.6 s for the CPU; the BLTC-lo calculation took 19.8 s and 12 iterations; assuming 0.23 s for each matrix-vector product, this implies 2.76 s for the GPU and 17.04 s for the CPU. This means that a large portion of the reported GPU run time was actually spent doing SLDM operations on 1 CPU core and hence a further reduction in run time could be gained by parallelizing those operations.

7. Summary

To account for hydrodynamic interactions among solvated molecules, Brownian dynamics simulations require correlated random displacements $\mathbf{g} = D^{1/2}\mathbf{z}$, where D is the $3N \times 3N$ RPY diffusion tensor for a system of N particles and $\mathbf{z}$ is a standard normal vector. The Spectral Lanczos Decomposition Method (SLDM) computes a sequence of approximations $\mathbf{g}_k \to \mathbf{g}$, but each step requires a dense matrix-vector product $D\mathbf{q}$ with a Lanczos vector $\mathbf{q}$, and the $O(N^2)$ cost of computing the product by direct summation (DS) is an obstacle for large-scale simulations. Here we employed the barycentric Lagrange treecode (BLTC) to reduce the cost of the matrix-vector product to $O(N \log N)$ while introducing a controllable approximation error. The BLTC is kernel-independent with relatively simple implementation and low memory requirements. Numerical experiments were performed to compare SLDM-BLTC with SLDM-DS for randomly distributed particles in a cube. The main results are as follows.

- The error E_k and increment I_k were computed for system size $N = 1e5$ and particle volume fraction PVF = 0.12, where the reference for the error was obtained by iterating SLDM-DS until I_k reached machine precision. In DS calculations, at a given step k the error E_k is an order of magnitude larger than the increment I_k (this was also previously seen for system size $N = 1e3$ [26]).
- In BLTC calculations with $N = 1e5$ and PVF = 0.12, two parameter sets were considered, BLTC-lo for low accuracy and BLTC-hi for high accuracy.
  - The BLTC-hi increment matches the DS increment down to machine precision, but the error saturates at 3-4 digit accuracy; this means that the BLTC-hi calculation converges, but the limit is different than in the DS calculation.
  - The BLTC-lo increment matches the DS and BLTC-hi increments for small k, but the error saturates at 1-2 digit accuracy while the increment continues decreasing to level 1e-4 after which it increases sharply and the calculation breaks down when the smallest eigenvalue of the Lanczos matrix T_k becomes negative.
  - Hence with proper choice of parameters the SLDM-BLTC can reduce the increment I_k to any desired level, but the error E_k may be substantially larger.
- SLDM-DS and SLDM-BLTC calculations were performed with PVF = 0.004, 0.04, 0.4 and system size $N = 1e5$. The iteration converges faster for small PVF and slower for large PVF. For all three PVFs, the BLTC-hi increment matches the DS increment down to machine precision, however the BLTC-lo increment matches the others down to machine precision only for small PVF = 0.004 and breaks down before reaching machine precision for larger PVF = 0.04, 0.4.
- The SLDM-DS and SLDM-BLTC run time and error were reported for single core calculations with convergence tolerance τ = 1e-2, 1e-3, 1e-4, 1e-5, system size $N = 1e5$ and PVF = 0.12. With proper choice of parameters, the BLTC error is close to the DS error while running faster.
- Single core serial calculations were compared with 32 core and single GPU parallel calculations for $N = 1e5, 1e6, 1e7$ and τ = 1e-2, 1e-4. Due to the expense of SLDM-DS, the reference solution and hence the error E_k could only be computed for $N = 1e5$.
  - For system size $N = 1e5$ and given tolerance τ, BLTC and DS have essentially the same error.
  - Where results are available, SLDM-DS and SLDM-BLTC require the same number of iterations to reach a given tolerance τ.
  - In most cases where results are available, the BLTC runs faster than DS, and the GPU runs faster than 32 cores; for example with $N = 1e6$ and $\tau = 1e-2$, DS-lo took 4355 s on 32 cores and 517 s on the GPU, while BLTC-lo took 52 s on 32 cores and 19.8 s on the GPU. This shows the efficiency gained by combining algorithmic acceleration (due to the BLTC) and hardware acceleration (due to the GPU).

Future research should examine the effect of approximation errors in the RPY matrix-vector product on the quality of BD simulations. In particular, since the present work considered only randomly distributed particles in a cube, the SLDM-BLTC should be tested for more realistic particle distributions representing polymers and biomolecules, especially where high accuracy may be needed as with intrinsically disordered proteins [41]. There are potentially several ways in which the correlated random displacements could be computed more efficiently, for example by preconditioning the SLDM [42–44] or applying a rational approximation of the matrix square root [45,46]. Further improvements of the GPU calculation could include parallelizing the SLDM operations that were performed here in serial and using an FMM extension of the BLTC on multiple GPUs [47,48].

While the present work dealt with RPY interactions in unbounded free space, another goal is to implement SLDM-BLTC with periodic boundary conditions. An earlier version of the treecode based on Taylor series (TTC) was used to compute Coulomb interactions in molecular dynamics simulations with periodic boundary conditions. Two approaches were considered; (1) the fundamental cell was replicated to one level of images and the particle-particle Coulomb interactions were computed using the TTC [49], (2) Ewald summation was applied where the real-space term involving the complementary error function was computed using the TTC, while the reciprocal-space term was computed directly [50]. However, the TTC computes the Taylor coefficients of the Coulomb potential recursively and this causes thread divergence on a GPU, while the barycentric Lagrange operations in the BLTC are kernel-independent and non-recursive and they can be done without thread divergence. Hence the BLTC could replace the TTC for computing periodic RPY interactions using either of these approaches. Other potential applications for SLDM-BLTC with periodic RPY interactions include the Positively Split Ewald method where the real-space and reciprocal-space terms are positive-definite [51,52], as well as hydrodynamic interactions in confined doubly-periodic geometry [53].

CRediT authorship contribution statement

Lei Wang: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation. **Robert Krasny:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Investigation, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors thank the reviewers for providing valuable comments that helped improve the article. This work was partially supported by National Science Foundation grant DMS-2110767.

Data availability

Data will be made available on request.

References

[1] T. Schlick, Molecular Modeling and Simulation: An Interdisciplinary Guide, Springer, New York, 2002.
[2] G.A. Huber, J.A. McCammon, Brownian dynamics simulations of biological molecules, Trends Chem. 1 (2019) 727–738.
[3] A. Muñiz-Chicharro, L.W. Votapka, R.E. Amaro, R.C. Wade, Brownian dynamics simulations of biomolecular diffusional association processes, WIREs Comput. Mol. Sci. 13 (2022) e1649.
[4] D.L. Ermak, J.A. McCammon, Brownian dynamics with hydrodynamic interactions, J. Chem. Phys. 69 (1978) 1352–1360.
[5] J. Rotne, S. Prager, Variational treatment of hydrodynamic interaction in polymers, J. Chem. Phys. 50 (1969) 4831–4837.
[6] H. Yamakawa, Transport properties of polymer chains in dilute solution: hydrodynamic interaction, J. Chem. Phys. 53 (1970) 436–443.
[7] J. Barnes, P. Hut, A hierarchical $O(N \log N)$ force-calculation algorithm, Nature 324 (1986) 446–449.
[8] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (1987) 325–348.
[9] A.-K. Tornberg, L. Greengard, A fast multipole method for the three-dimensional Stokes equations, J. Comput. Phys. 227 (2008) 1613–1619.
[10] F. Bülow, P. Hamberger, H. Nirschl, W. Dörfler, A scalable parallel Stokesian dynamics method for the simulation of colloidal suspensions, Comput. Phys. Commun. 204 (2016) 107–120.
[11] L. Wang, S. Tlupova, R. Krasny, A treecode algorithm for 3D stokeslets and stresslets, Adv. Appl. Math. Mech. 11 (2019) 737–756.
[12] W. Guan, X. Geng, J. Huang, G. Huber, W. Li, J. McCammon, B. Zhang, RPYFMM: parallel adaptive fast multipole method for Rotne-Prager-Yamakawa tensor in biomolecular hydrodynamics, Comput. Phys. Commun. 227 (2018) 99–108.
[13] L. Ying, G. Biros, D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, J. Comput. Phys. 196 (2004) 591–626.
[14] W. Fong, E. Darve, The black-box fast multipole method, J. Comput. Phys. 228 (2009) 8712–8725.
[15] W. Hackbusch, B.N. Khoromskij, S. Sauter, On $\mathcal{H}^2$-matrices, Lect. Appl. Math. (2000) 9–29.
[16] W. Hackbusch, S. Börm, Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices, Computing 69 (2002) 1–35.
[17] Z. Liang, Z. Gimbutas, L. Greengard, J. Huang, S. Jiang, A fast multipole method for the Rotne-Prager-Yamakawa tensor and its applications, J. Comput. Phys. 234 (2013) 133–139.
[18] X. Xing, H. Huang, E. Chow, A hierarchical matrix approach for computing hydrodynamic interactions, J. Comput. Phys. 448 (2022) 110761.
[19] D. Petera, M. Muthukumar, Brownian dynamics simulation of bead-rod chains under shear with hydrodynamic interaction, J. Chem. Phys. 111 (1999) 7614–7623.
[20] R.G. Larson, Principles for coarse-graining polymer molecules in simulations of polymer fluid mechanics, Mol. Phys. 102 (2004) 341–351.
[21] M. Fixman, Construction of Langevin forces in the simulation of hydrodynamic interaction, Macromolecules 19 (1986) 1204–1207.
[22] S. Jiang, Z. Liang, J. Huang, A fast algorithm for Brownian dynamics simulation with hydrodynamic interactions, Math. Comput. 82 (2013) 1631–1645.

[23] R.M. Jendrejack, M.D. Graham, J.J. de Pablo, Hydrodynamic interactions in long chain polymers: application of the Chebyshev polynomial approximation in stochastic simulations, J. Chem. Phys. 113 (2000) 2894.
[24] Y. Saad, Analysis of some Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal. 29 (1992) 209–228.
[25] V. Druskin, L. Knizhnerman, Krylov subspace approximation of eigenpairs and matrix functions in exact and computer arithmetic, Numer. Linear Algebra Appl. 2 (1995) 205–217.
[26] T. Ando, E. Chow, Y. Saad, J. Skolnick, Krylov subspace methods for computing hydrodynamic interactions in Brownian dynamics simulation, J. Chem. Phys. 137 (2012) 064160.
[27] T. Geyer, U. Winter, An N^2 approximation for hydrodynamic interactions in Brownian dynamics simulations, J. Chem. Phys. 130 (2009) 114905.
[28] T. Geyer, Many-particle Brownian and Langevin dynamics simulations with the Brownmove package, BMC Biophys. 4 (2011) 7.
[29] A.H. Elcock, Molecule-centered method for accelerating the calculation of hydrodynamic interactions in Brownian dynamics simulations containing many flexible biomolecules, J. Chem. Theory Comput. 9 (2013) 3224–3239.
[30] J.-P. Berrut, L.N. Trefethen, Barycentric Lagrange interpolation, SIAM Rev. 46 (2004) 501–517.
[31] L. Wang, R. Krasny, S. Tlupova, A kernel-independent treecode based on barycentric Lagrange interpolation, Commun. Comput. Phys. 28 (2020) 1415–1436.
[32] E. Wajnryb, K.A. Mizerski, P.J. Zuk, P. Szymczak, Generalization of the Rotne-Prager-Yamakawa mobility and shear disturbance tensors, J. Fluid Mech. 731 (2013) 1–12.
[33] L. Wang, A kernel-independent treecode for general regularized Rotne-Prager-Yamakawa tensor, Adv. Appl. Math. Mech. 13 (2021) 296–310.
[34] J.W. Demmel, Applied Numerical Linear Algebra, SIAM, 1997.
[35] H.A. Boateng, R. Krasny, Comparison of treecodes for computing electrostatic potentials in charged particle systems with disjoint targets and sources, J. Comput. Chem. 34 (2013) 2159–2167.
[36] H.E. Salzer, Lagrangian interpolation at the Chebyshev points $x_{n,\nu} \equiv \cos(\nu\pi/n), \nu = 0(1)n$; some unnoted advantages, Comput. J. 15 (1972) 156–159.
[37] L.N. Trefethen, Approximation Theory and Approximation Practice, extended edition, SIAM, 2019.
[38] N.J. Higham, The numerical stability of barycentric Lagrange interpolation, IMA J. Numer. Anal. 24 (2004) 547–556.
[39] S.B. Zimmerman, S.O. Trach, Estimation of macromolecule concentrations and excluded volume effects for the cytoplasm of Escherichia coli, J. Mol. Biol. 222 (1991) 599–620.
[40] L. Wang, SLDM code, https://www.github.com/Treecodes/stokes-treecode, 2024.
[41] S.-H. Ahn, G.A. Huber, J.A. McCammon, Investigating intrinsically disordered proteins with Brownian dynamics, Front. Mol. Biosci. 9 (2022) 898838.
[42] T. Ando, E. Chow, J. Skolnick, Dynamic simulation of concentrated macromolecular solutions with screened long-range hydrodynamic interactions: algorithm and limitations, J. Chem. Phys. 139 (2013).
[43] E. Chow, Y. Saad, Preconditioned Krylov subspace methods for sampling multivariate Gaussian distributions, SIAM J. Sci. Comput. 36 (2014) A588–A608.
[44] J. Chen, W. Geng, On preconditioning the treecode-accelerated boundary integral (TABI) Poisson-Boltzmann solver, J. Comput. Phys. 373 (2018) 750–762.
[45] N. Hale, N.J. Higham, L.N. Trefethen, Computing $A^{\alpha}, \log(A)$, and related matrix functions by contour integrals, SIAM J. Numer. Anal. 46 (2008) 2505–2523.
[46] T. Chen, A. Greenbaum, C. Musco, C. Musco, Low-memory Krylov subspace methods for optimal rational matrix function approximation, SIAM J. Matrix Anal. Appl. 44 (2023) 670–692.
[47] N. Vaughn, L. Wilson, R. Krasny, A GPU-accelerated barycentric Lagrange treecode, in: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2020, pp. 701–710.
[48] L. Wilson, N. Vaughn, R. Krasny, A GPU-accelerated fast multipole method based on barycentric Lagrange interpolation and dual tree traversal, Comput. Phys. Commun. 265 (2021) 108017.
[49] H.A. Boateng, Periodic Coulomb tree method: an alternative to parallel particle Mesh Ewald, J. Chem. Theory Comput. 16 (2019) 7–17.
[50] Z.-H. Duan, R. Krasny, An Ewald summation based multipole method, J. Chem. Phys. 113 (2000) 3492–3495.
[51] D. Lindbo, A.-K. Tornberg, Spectrally accurate fast summation for periodic Stokes potentials, J. Comput. Phys. 229 (2010) 8994–9010.
[52] A.M. Fiore, F. Balboa Usabiaga, A. Donev, J.W. Swan, Rapid sampling of stochastic displacements in Brownian dynamics simulations, J. Chem. Phys. 146 (2017) 124116.
[53] A. Hashemi, R.P. Peláez, S. Natesh, B. Sprinkle, O. Maxian, Z. Gan, A. Donev, Computing hydrodynamic interactions in confined doubly periodic geometries in linear time, J. Chem. Phys. 158 (2023) 154101.