Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2019-0177

Vol. **28**, No. 4, pp. 1415-1436
October 2020

A Kernel-Independent Treecode Based on Barycentric Lagrange Interpolation

Lei Wang[1], Robert Krasny[2,*] and Svetlana Tlupova[3]

[1] *Department of Mathematical Sciences, University of Wisconsin-Milwaukee, Milwaukee, WI 53201, USA.*
[2] *Department of Mathematics, University of Michigan, Ann Arbor, MI 48109, USA.*
[3] *Department of Mathematics, Farmingdale State College, Farmingdale, NY 11735, USA.*

Received 20 October 2019; Accepted (in revised version) 3 February 2020

Abstract. A kernel-independent treecode (KITC) is presented for fast summation of particle interactions. The method employs barycentric Lagrange interpolation at Chebyshev points to approximate well-separated particle-cluster interactions. The KITC requires only kernel evaluations, is suitable for non-oscillatory kernels, and relies on the scale-invariance property of barycentric Lagrange interpolation. For a given level of accuracy, the treecode reduces the operation count for pairwise interactions from $\mathcal{O}(N^2)$ to $\mathcal{O}(N\log N)$, where N is the number of particles in the system. The algorithm is demonstrated for systems of regularized Stokeslets and rotlets in 3D, and numerical results show the treecode performance in terms of error, CPU time, and memory consumption. The KITC is a relatively simple algorithm with low memory consumption, and this enables a straightforward OpenMP parallelization.

AMS subject classifications: 65D99, 76D07

Key words: Treecode, barycentric Lagrange interpolation, scale-invariance, Chebyshev points, regularized Stokeslets.

1 Introduction

Consider the problem of evaluating the sum

$$u(\mathbf{x}_i)=\sum_{j=1}^{N}k(\mathbf{x}_i,\mathbf{x}_j)f_j,\quad i=1,\cdots,N, \tag{1.1}$$

*Corresponding author. *Email addresses:* wang256@uwm.edu (L. Wang), krasny@umich.edu (R. Krasny), tlupovs@farmingdale.edu (S. Tlupova)

 ©2020 Global-Science Press

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

where $u(\mathbf{x}_i)$ is a velocity (or potential or force) and $\{\mathbf{x}_i\} \subset \mathbb{R}^d$ is a set of particles with weights $\{f_i\}$. Depending on the application, the velocity and weights may be scalars or vectors, and the kernel may be a tensor. The kernel $k(\mathbf{x},\mathbf{y})$ describes the interaction between a target particle $\mathbf{x}$ and a source particle $\mathbf{y}$, and we are interested in non-oscillatory kernels that are smooth for $\mathbf{x} \neq \mathbf{y}$ and decay slowly for $|\mathbf{x}-\mathbf{y}| \to \infty$. It is understood that if the kernel is singular for $\mathbf{x} = \mathbf{y}$, then the sum omits the $i = j$ term.

These sums arise in particle simulations involving point masses, point charges, and point vortices, as well as in boundary element methods where the particles are quadrature points. Evaluating (1.1) by direct summation requires $\mathcal{O}(N^2)$ operations which is prohibitively expensive when N is large, and several fast methods have been developed to reduce the cost. One can distinguish between two types of methods, *particle-mesh methods* in which the particles are projected onto a uniform mesh where the FFT or multigrid can be used (e.g. P3M [29], particle-mesh Ewald [16], spectral Ewald [2], multilevel summation [9, 27]), and *tree-based methods* in which the particles are partitioned into a hierarchy of clusters with a tree structure and the particle-particle interactions are replaced by particle-cluster or cluster-cluster approximations (e.g. treecode [4], fast multipole method (FMM) [24], panel clustering [26]).

Tree-based methods. The present work is concerned with tree-based methods that rely on degenerate kernel approximations of the form,

$$k(\mathbf{x},\mathbf{y}) \approx \sum_{k=0}^{n} \phi_k(\mathbf{x})\psi_k(\mathbf{y}). \tag{1.2}$$

Such approximations can be classified as *near-field/local* or *far-field/multipole* depending on their domain of validity in the variables $\mathbf{x},\mathbf{y}$. The treecode originally used a far-field monopole approximation for the Newtonian potential [4], while the FMM improved on this by employing higher-order multipole and local approximations, in particular using Laurent series for the 2D Laplace kernel and spherical harmonics for the 3D Laplace kernel [24, 25]. Later versions of the FMM used plane wave expansions for the 3D Laplace kernel [10] and spherical Bessel function expansions for the Yukawa potential [23]. Methods based on Cartesian Taylor expansions were also developed for some common kernels [13, 15, 30, 36, 38, 52, 57].

Kernel-independent methods. The tree-based methods cited above rely on analytic series expansions specific to each kernel and alternative approximation methods have been investigated. An early example in this direction was an FMM for Laplace kernels based on discretizing the Poisson integral formula [3], and this was followed by a pseudoparticle method that reproduces the multipole moments for these kernels [39]. Later work developed approximations suitable for a wide class of non-oscillatory kernels. One approach based on polynomial interpolation [32, section 11.4] has been applied in the context of multilevel approximation [21], hierarchical matrices [7], and the black-box FMM (bbFMM) [19]. An alternative method employed in the kernel-independent FMM (KIFMM) uses equivalent densities [60, 61], while other kernel-independent FMMs

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

use Legendre expansions [22], matrix compression based on skeletonization [42], and truncated Fourier series [62]. Recently an FMM based on the Cauchy integral formula and Laplace transform was proposed for general analytic functions [35], and a kernel-independent treecode was developed using approximate skeletonization for particle systems in high dimensions [41].

Present work. There is ongoing interest in exploring different strategies for fast summation of particle interactions, and the present work contributes a kernel-independent treecode (KITC) with operation count $\mathcal{O}(N\log N)$ in which the far-field approximation uses barycentric Lagrange interpolation at Chebyshev points [6,56]. The barycentric Lagrange interpolant can be efficiently implemented and has good stability properties [28, 43,47]; the 1D case is reviewed in [6,56] and here we apply it in 3D using a tensor product to compute well-separated particle-cluster approximations.

It should be noted that the bbFMM [19] and KITC both use polynomial interpolation, but they differ in two ways. The first difference concerns the interpolating polynomial; the bbFMM uses Chebyshev points of the 1st kind (roots of Chebyshev polynomials) and expresses the Lagrange polynomials in terms of Chebyshev polynomials, while the KITC uses Chebyshev points of the 2nd kind (extrema of Chebyshev polynomials) and expresses the Lagrange polynomials in barycentric form; as explained in Section 3, this enables the KITC to take advantage of the scale-invariance property of barycentric Lagrange interpolation. The second difference concerns the algorithm structure; the KITC uses only far-field approximations and avoids the multipole-to-local translations and SVD compression steps in the bbFMM [19]. With these choices the KITC is a relatively simple algorithm with low memory consumption, and this enables a straightforward OpenMP parallelization.

We will present numerical results motivated by the method of regularized Stokeslets (MRS) for slow viscous flow [11, 12]. The MRS has been applied to simulate cilia- and flagella-driven flow [18,50], helical swimming [12], slender body flow [8], coupled Stokes-Darcy flow [53], and flow around elastic rods [45]. Due to the complexity of the MRS kernels, they are prime candidates for kernel-independent fast summation methods, but as far as we know only recently has the KIFMM been applied to MRS simulations [33,48]. Here we apply the KITC to systems of regularized Stokeslets and rotlets from [48], and the results demonstrate the method's good performance in terms of accuracy, efficiency, and memory consumption in serial and parallel simulations.

The paper is organized as follows. Section 2 discusses polynomial interpolation and its application to kernel approximation. Section 3 reviews barycentric Lagrange interpolation following [6, 56]. Section 4 explains how the interpolant is used to approximate particle-cluster interactions. Section 5 describes how some quantities called modified weights are computed. Section 6 presents the KITC algorithm. Section 7 reviews the MRS kernels (regularized Stokeslet and rotlet). Section 8 presents numerical results for two examples motivated by recent MRS simulations [48]. A summary is given in Section 9.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

2 Polynomial interpolation and kernel approximation

We begin by recalling some basic facts about polynomial interpolation in 1D [56]. Given a function $f(t)$ and $n+1$ distinct points $s_k \in [-1,1]$ for $k=0,\cdots,n$, there is a unique polynomial $p_n(t)$ of degree at most n that interpolates the function at these points, $p_n(s_k)=f(s_k)$, $k=0,\cdots,n$. The Lagrange form of the interpolating polynomial is

$$p_n(t) = \sum_{k=0}^{n} f_k L_k(t), \quad f_k = f(s_k), \quad k=0,\cdots,n, \tag{2.1}$$

where the Lagrange polynomials,

$$L_k(t) = \frac{\Pi_{k'=0,k'\neq k}^{n}(t-s_{k'})}{\Pi_{k'=0,k'\neq k}^{n}(s_k-s_{k'})}, \quad k=0,\cdots,n, \tag{2.2}$$

have degree n and satisfy $L_k(s_{k'})=\delta_{kk'}$. We view the interpolating polynomial $p_n(t)$ as an approximation to $f(t)$, and applying this idea to a kernel $k(x,y)$ in 1D, we hold x fixed and interpolate with respect to y to obtain

$$k(x,y) \approx \sum_{k=0}^{n} k(x,s_k) L_k(y). \tag{2.3}$$

The approximation on the right is a polynomial of degree n in the variable y and it interpolates the kernel at $y=s_k$; this idea is well known (e.g. [32, section 11.4]).

Now consider a kernel $k(\mathbf{x},\mathbf{y})$ in 3D and a tensor product set of grid points $\mathbf{s_k} = (s_{k_1},s_{k_2},s_{k_3}) \in [-1,1]^3$, where $\mathbf{k}=(k_1,k_2,k_3)$ is a multi-index with $k_\ell = 0,\cdots,n$ for $\ell=1,2,3$. As above we hold $\mathbf{x}$ fixed and interpolate with respect to $\mathbf{y}=(y_1,y_2,y_3)$ to obtain

$$k(\mathbf{x},\mathbf{y}) \approx \sum_{k_1=0}^{n}\sum_{k_2=0}^{n}\sum_{k_3=0}^{n} k(\mathbf{x},\mathbf{s_k}) L_{k_1}(y_1) L_{k_2}(y_2) L_{k_3}(y_3). \tag{2.4}$$

In this case the approximation on the right is a polynomial of degree n in each variable (y_1,y_2,y_3) and it interpolates the kernel at the grid points $\mathbf{y}=\mathbf{s_k}$. Moreover, the Lagrange polynomial expressions (2.3) and (2.4) are degenerate kernel approximations of the form (1.2).

3 Barycentric Lagrange interpolation

The expression for the interpolating polynomial described in Section 2 is not well-suited for practical computing due to cost and stability issues; the problem though is not with the Lagrange form (2.1) for $p_n(t)$, but rather with the expression (2.2) for the Lagrange

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

polynomials $L_k(t)$. Berrut and Trefethen [6] advocated using instead the 2nd barycentric form of the Lagrange polynomials,

$$L_k(t)=\frac{\dfrac{w_k}{t-s_k}}{\displaystyle\sum_{k'=0}^{n}\frac{w_{k'}}{t-s_{k'}}},\quad w_k=\frac{1}{\Pi_{k'=0,k'\neq k}^{n}(s_k-s_{k'})},\quad k=0,\cdots,n, \tag{3.1}$$

where w_k are the barycentric weights. This form is mathematically equivalent to (2.2), with the understanding that the removable singularity at $t=s_k$ is resolved by setting $L_k(s_{k'})=\delta_{kk'}$; to enforce this condition, following [6] the code is written so that if the argument t is closer to an interpolation point s_k than some tolerance, this is flagged and the correct value $L_k(s_{k'})=\delta_{kk'}$ is provided. In our implementation the tolerance is the minimum positive IEEE double precision floating point number, DBL_MIN= 2.22507e-308; further details will be given in Section 5.

We work with Chebyshev points of the 2nd kind,

$$s_k=\cos\theta_k,\quad \theta_k=\frac{k\pi}{n},\quad k=0,\cdots,n. \tag{3.2}$$

In this case the interpolating polynomial $p_n(t)$ converges rapidly and uniformly on $[-1,1]$ to $f(t)$ as n increases, under mild smoothness assumptions on the given function [56]. Note that computing the barycentric weights w_k by the definition (3.1) requires $\mathcal{O}(n^2)$ operations, but this expense disappears for the s_k chosen in (3.2) because in that case the following simple weights can be used instead [49,56],

$$w_k=(-1)^k\delta_k,\quad \delta_k=\begin{cases}1/2, & k=0 \text{ or } k=n,\\ 1, & k=1,\cdots,n-1.\end{cases} \tag{3.3}$$

This relies on a scale-invariance property of the barycentric form of $L_k(t)$ in (3.1); namely, if the weights w_k have a common constant factor $\alpha\neq 0$, then α can be cancelled from the numerator and denominator, and (3.1) stays the same.

Hence the barycentric Lagrange form of the interpolating polynomial is

$$p_n(t)=\frac{\displaystyle\sum_{k=0}^{n}\frac{w_k}{t-s_k}f_k}{\displaystyle\sum_{k=0}^{n}\frac{w_k}{t-s_k}}, \tag{3.4}$$

and with the simple weights (3.3), evaluating $p_n(t)$ requires $\mathcal{O}(n)$ operations [6,56].

The scale-invariance property is also important when working on different intervals; if $[-1,1]$ is linearly mapped to $[a,b]$ by $t\to\frac{1}{2}(a+b+t(b-a))$, then the weights w_k defined in (3.1) gain a factor of $2^n/(b-a)^n$ which could lead to overflow or underflow, but

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

this factor can be safely omitted due to scale-invariance. This means that the simple weights (3.3) can be used for any interval $[a,b]$, along with the linearly mapped Chebyshev points; this is important in the present work because the treecode uses intervals of different sizes. Note also that barycentric Lagrange interpolation is stable in finite precision arithmetic [28, 43, 47], and the Chebfun software package uses this form of polynomial interpolation [14].

For comparison, the bbFMM [19] uses Chebyshev points of the 1st kind,

$$\bar{s}_k = \cos\bar{\theta}_k, \quad \bar{\theta}_k = \frac{(2k-1)\pi}{2n}, \quad k=1,\cdots,n, \tag{3.5}$$

and a Chebyshev Lagrange form of the interpolating polynomial,

$$p_{n-1}(t) = \sum_{k=1}^{n} f_k \bar{L}_k(t), \quad \bar{L}_k(t) = \frac{1}{n} + \frac{2}{n}\sum_{k'=1}^{n-1} T_{k'}(\bar{s}_k)T_{k'}(t), \quad k=1,\cdots,n, \tag{3.6}$$

where $T_k(t)$ is the kth degree Chebyshev polynomial and $\bar{L}_k(\bar{s}_{k'}) = \delta_{kk'}$. The cost of evaluating $p_{n-1}(t)$ by directly summing (3.6) is $\mathcal{O}(n^2)$, or $\mathcal{O}(n\log n)$ if a fast transform is used. We are not aware of an analog of the barycentric scale-invariance property.

4 Particle-cluster interactions

In a treecode the particles $\{\mathbf{x}_i\}$ are partitioned into a hierarchy of clusters $\{C\}$, and the sum (1.1) is written as

$$u(\mathbf{x}_i) = \sum_{j=1}^{N} k(\mathbf{x}_i,\mathbf{x}_j) f_j = \sum_{C} u(\mathbf{x}_i, C), \quad i=1,\cdots,N, \tag{4.1}$$

where

$$u(\mathbf{x}_i, C) = \sum_{\mathbf{y}_j \in C} k(\mathbf{x}_i,\mathbf{y}_j) f_j, \tag{4.2}$$

is the interaction between a target particle $\mathbf{x}_i$ and a source cluster $C = \{\mathbf{y}_j\}$. The sum over C in (4.1) denotes a suitable subset of clusters depending on the target particle. Fig. 1a depicts a particle-cluster interaction with target particle $\mathbf{x}_i$, cluster center $\mathbf{y}_c$, cluster radius r, and particle-cluster distance $R = |\mathbf{x}_i - \mathbf{y}_c|$. In this work the clusters are rectangular boxes whose sides are aligned with the coordinate axes, and the cluster radius r is the half-length of the box diagonal. The interpolation scheme easily handles rectangular boxes because it uses a tensor product in the x,y,z directions and the Chebyshev points are mapped to the box appropriately in each direction. When the particle and cluster are well-separated (the criterion is given in Section 6), the interaction (4.2) is computed using the kernel approximation (2.4); this is depicted in Fig. 1b showing the Chebyshev grid points $\mathbf{s_k}$ in the cluster. Hence the result is that the target particle $\mathbf{x}_i$ interacts with the interpolation points $\mathbf{s_k}$ rather than the source particles $\mathbf{y}_j$.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

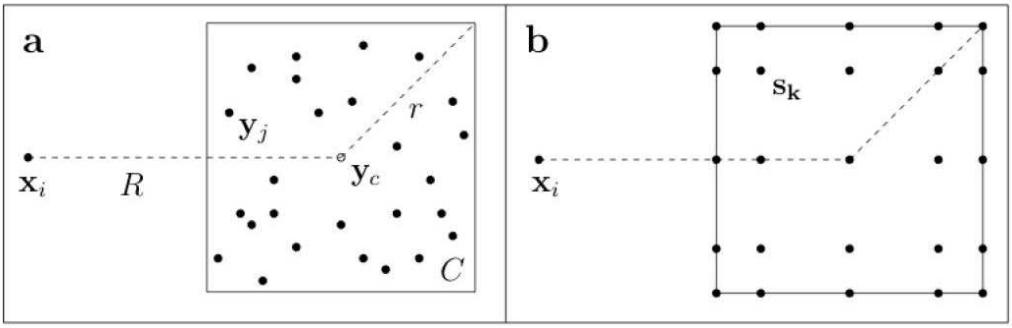

Figure 1: Particle-cluster interaction, (a) target particle $\mathbf{x}_i$, source cluster $C=\{\mathbf{y}_j\}$, center $\mathbf{y}_c$, radius r, particle-cluster distance $R=|\mathbf{x}_i-\mathbf{y}_c|$, (b) Chebyshev grid points $\mathbf{s_k}$ in cluster C.

In more detail, we substitute the kernel approximation (2.4) into the particle-cluster interaction (4.2) and switch the order of summation to obtain the far-field particle-cluster approximation,

$$u(\mathbf{x}_i,C)\approx\sum_{k_1=0}^{n}\sum_{k_2=0}^{n}\sum_{k_3=0}^{n}k(\mathbf{x}_i,\mathbf{s_k})\widehat{f}_{\mathbf{k}}, \tag{4.3}$$

where the modified weights are

$$\widehat{f}_{\mathbf{k}}=\sum_{\mathbf{y}_j\in C}L_{k_1}(y_{j1})L_{k_2}(y_{j2})L_{k_3}(y_{j3})f_j,\quad k_1,k_2,k_3=0,\cdots,n, \tag{4.4}$$

and $\mathbf{y}_j=(y_{j1},y_{j2},y_{j3})$. Note that when the target $\mathbf{x}_i$ is well-separated from the sources $\mathbf{y}_j$, the kernel $k(\mathbf{x},\mathbf{y})$ is interpolated on a subdomain where it is smooth and this ensures the accuracy of the approximation.

We see that (4.3) has the form of a particle-cluster interaction (4.2), where the source particles and weights $\{\mathbf{y}_j,f_j\}$ are replaced by the Chebyshev grid points and modified weights $\{\mathbf{s_k},\widehat{f}_{\mathbf{k}}\}$. This is advantageous for two reasons. First, the modified weights $\widehat{f}_{\mathbf{k}}$ for a given cluster are independent of the target particle $\mathbf{x}_i$, so they can be precomputed and re-used for different targets; details are given below. Second, the cost of the direct sum (4.2) is $\mathcal{O}(N_c)$, where N_c is the number of particles in cluster C, while the cost of the approximation (4.3) is $\mathcal{O}(n^3)$ assuming the modified weights are known, so there is a cost reduction when $N_c\gg n^3$. Finally note that the approximation (4.3) depends only on kernel evaluations and hence qualifies as kernel-independent.

5 Computation of modified weights

Algorithm 1 describes how the modified weights are computed. Before proceeding, note that the Lagrange polynomial terms in (4.4) in barycentric form are

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

$$L_{k_\ell}(y_{j\ell}) = \frac{\dfrac{w_{k_\ell}}{y_{j\ell}-s_{k_\ell}}}{\displaystyle\sum_{k'_\ell=0}^{n} \frac{w_{k'_\ell}}{y_{j\ell}-s_{k'_\ell}}} = \frac{a_{j,\ell,k_\ell}}{\displaystyle\sum_{k'_\ell=0}^{n} a_{j,\ell,k'_\ell}}, \quad a_{j,\ell,k_\ell} = \frac{w_{k_\ell}}{y_{jl}-s_{k_\ell}}, \tag{5.1}$$

where the variables a_{j,ℓ,k_ℓ} are introduced for clarity. There are three loops, the outer loop over source particles $j = 1,\cdots,N_c$, the 1st inner loop over coordinate indices $\ell = 1,2,3$, and the 2nd inner loop over Chebyshev points $k_\ell = 0,\cdots,n$. To handle the removable singularity at $y_{j\ell} = s_{k_\ell}$, we adapt the procedure in [6, page 510].

In Algorithm 1, the source particles and weights $\mathbf{y}_j, f_j$ for a given cluster are input, and the modified weights $\widehat{f}_{\mathbf{k}}$ are output. The modified weights are initialized to zero on line 4, the loop over source particles starts on line 6, and the flags and sums are initialized on line 7. Lines 9-13 loop over the coordinate indices and Chebyshev points to compute the terms a_{j,ℓ,k_ℓ}; this requires $\mathcal{O}(nN_c)$ operations, and a temporary storage array of size $\mathcal{O}(nN_c)$ for the terms a_{j,ℓ,k_ℓ}, which is reused for different clusters. Line 10 checks whether a source particle coordinate is close to a Chebyshev point; if so, then a flag records the index. Then on lines 15-17 the code checks to see whether a flag was set; if so, then the temporary variables are adjusted to enforce the condition $L_k(s_{k'}) = \delta_{kk'}$. Then on lines 20-22 the code loops over the tensor product of Chebyshev point indices and increments the modified weights $\widehat{f}_{\mathbf{k}}$ as indicated in (4.4); this requires $\mathcal{O}(n^3 N_c)$ operations and $\mathcal{O}(n^3)$ storage for each cluster. The modified weights are computed this way for each cluster when the tree is constructed and in practice this requires only a small fraction of the total KITC CPU time; for example in one case with N = 640 K particles, the total CPU time was 149 s, while computing the modified weights took less than 2 s.

The quantities $L_{k_\ell}(y_{j\ell})$ could be computed instead using the Chebyshev form (3.6), but the argument of the Chebyshev polynomials appearing there lies in the unit interval $[-1,1]$, so the particles $\mathbf{y}_j$ must be mapped to the unit cube $[-1,1]^3$, with cost $\mathcal{O}(N_c)$. The barycentric form (3.1) does not require this mapping; the particles stay in place in the cluster; hence the barycentric form has an advantage over the Chebyshev form since it avoids the cost of mapping the particles $\mathbf{y}_j$.

6 Kernel-independent treecode algorithm

Aside from using barycentric Lagrange interpolation for the far-field approximation, the present algorithm is similar to previous treecodes based on analytic series expansions [4, 13, 36]. The procedure is outlined in Algorithm 2. After inputting the particle data and treecode parameters, a hierarchical tree of particle clusters is built and the modified weights for each cluster are computed. In our implementation the root cluster of the tree is the smallest rectangular box enclosing the particles. The root is bisected in any coordinate direction for which the side length is greater than $\ell_{\max}/\sqrt{2}$, where $\ell_{\max}$ is the

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

Algorithm 1 computation of modified weights in (4.4)

1: input: source particles and weights for a given cluster, $\mathbf{y}_j, f_j$
2: input: Chebyshev points mapped to the cluster, $s_{\mathbf{k}}$
3: output: modified weights, $\widehat{f}_{\mathbf{k}}$
4: initialize all $\widehat{f}_{\mathbf{k}} = \widehat{f}(k_1,k_2,k_3) = 0$
5: % loop over source particles
6: for $j = 1:N_c$
7: initialize flag(1:3) = -1, sum(1:3) = 0
8: % loop over coordinate indices and Chebyshev points to compute a_{j,ℓ,k_ℓ}
9: for $\ell = 1:3$ and $k_\ell = 0:n$
10: if $|y_{j\ell} - s_{k_\ell}| \le$ DBL_MIN, flag$(\ell) = k_\ell$
11: else $a(j,\ell,k_\ell) = w_{k_\ell}/(y_{j\ell} - s_{k_\ell})$, sum$(\ell)$ += $a(j,\ell,k_\ell)$
12: end if
13: end for
14: % if a flag was set, adjust sum(ℓ) and $a(j,\ell,k_\ell)$ to handle removable singularity
15: for $\ell = 1:3$
16: if flag$(\ell) > -1$, sum$(\ell) = 1$, $a(j,\ell,0:n) = 0$, $a(j,\ell,$flag$(\ell)) = 1$, end if
17: end for
18: denom = sum(1) $\cdot$ sum(2) $\cdot$ sum(3)
19: % loop over tensor product of Chebyshev point indices as in (4.4)
20: for $(k_1,k_2,k_3) = (0:n,0:n,0:n)$
21: $\widehat{f}(k_1,k_2,k_3)$ += $(a(j,1,k_1) \cdot a(j,2,k_2) \cdot a(j,3,k_3)/\text{denom}) \cdot f_j$
22: end for
23: end for

maximum side length of the root. Hence the root is bisected in its long directions, but not its short directions, and the child clusters are bisected in the same way. The process continues until a cluster has fewer than N_0 particles, a user-specified parameter. The clusters obtained this way are rectangular boxes, and a cluster may have 8, 4, or 2 children depending on which sides were bisected. The flexibility to use rectangular boxes instead of cubes is important in cases like Example 2 below where the particles lie in a rectangular slab.

Turning to Algorithm 2, after building the tree of clusters (line 5), the code cycles through the particles (line 7), and each particle interacts with clusters, starting at the root and proceeding to the child clusters. The multipole acceptance criterion (MAC),

$$\frac{r}{R} \le \theta, \tag{6.1}$$

determines whether a particle $\mathbf{x}_i$ and cluster C are well-separated, where (recalling Fig. 1) r is the cluster radius, R is the particle-cluster distance, and θ is a user-specified parameter. If the MAC (6.1) is satisfied, the particle-cluster interaction is computed by barycentric Lagrange interpolation (4.3) with user-specified degree n; otherwise the code checks

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

Algorithm 2 kernel-independent treecode

```
input: particle coordinates and weights x_i, f_i, i = 1, ..., N
input: treecode MAC parameter θ, polynomial degree n, maximum leaf size N_0
output: particle velocities u_i, i = 1, ..., N
program main
    build tree of particle clusters
    compute modified weights f̂_k in (4.4) for each cluster
    for i = 1, ..., N, compute_velocity(x_i, root), end for
end program
subroutine compute_velocity(x, C)
    if MAC is satisfied
        compute particle-cluster interaction by approximation (4.3)
    else
        if C is a leaf, compute particle-cluster interaction by direct sum (4.2)
    else
        for each child C' of C, compute_velocity(x, C'), end for
end subroutine
```

the child clusters, or if the cluster is a leaf (no children), then the interaction is computed directly by (4.2).

This is essentially the Barnes-Hut algorithm [4], extended to higher-order particle-cluster approximations computed by barycentric Lagrange interpolation. There are two stages; stage 1 is building the tree and computing the modified weights for each cluster, and stage 2 is computing the particle-cluster interactions. In principle both stages scale like $\mathcal{O}(N\log N)$, although in practice stage 1 is much faster than stage 2. There are three user-specified parameters (θ,n,N_0) that control the accuracy and efficiency of the treecode; the present work uses representative values with no claim that they are optimal.

7 Regularized Stokes kernels

The method of regularized Stokeslets (MRS) uses regularized kernels to represent point forces and torques in slow viscous flow [11, 12, 45]. In particular, consider the following functions defined on $\mathbb{R}^3$,

$$H_1(r)=\frac{2\epsilon^2+r^2}{8\pi(r^2+\epsilon^2)^{3/2}},\quad H_2(r)=\frac{1}{8\pi(r^2+\epsilon^2)^{3/2}},\quad Q(r)=\frac{5\epsilon^2+2r^2}{8\pi(r^2+\epsilon^2)^{5/2}},\tag{7.1a}$$

$$D_1(r)=\frac{10\epsilon^4-7\epsilon^2r^2-2r^4}{8\pi(r^2+\epsilon^2)^{7/2}},\quad D_2(r)=\frac{21\epsilon^2+6r^2}{8\pi(r^2+\epsilon^2)^{7/2}},\tag{7.1b}$$

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

where $r=|\mathbf{x}-\mathbf{y}|$, $\mathbf{x}$ is a target, $\mathbf{y}$ is a source, and ϵ is the MRS regularization parameter. Given a set of source particles $\{\mathbf{y}_j\}$ with forces $\{\mathbf{f}_j\}$ and torques $\{\mathbf{n}_j\}$, for $j=1,\cdots,N$, the velocity induced by regularized Stokeslets is

$$\mathbf{u}(\mathbf{x})=\sum_{j=1}^{N}\left(\mathbf{f}_j H_1(r_j)+\left[\mathbf{f}_j\cdot(\mathbf{x}-\mathbf{y}_j)\right](\mathbf{x}-\mathbf{y}_j)H_2(r_j)\right), \tag{7.2}$$

where $r_j=|\mathbf{x}-\mathbf{y}_j|$, while the linear velocity and angular velocity induced by regularized Stokeslets and rotlets are

$$\mathbf{u}(\mathbf{x})=\sum_{j=1}^{N}\left(\mathbf{f}_j H_1(r_j)+\left[\mathbf{f}_j\cdot(\mathbf{x}-\mathbf{y}_j)\right](\mathbf{x}-\mathbf{y}_j)H_2(r_j)+\frac{1}{2}\left[\mathbf{n}_j\times(\mathbf{x}-\mathbf{y}_j)\right]Q(r_j)\right), \tag{7.3a}$$

$$\mathbf{w}(\mathbf{x})=\sum_{j=1}^{N}\left(\frac{1}{2}\left[\mathbf{f}_j\times(\mathbf{x}-\mathbf{y}_j)\right]Q(r_j)+\frac{1}{4}\mathbf{n}_j D_1(r_j)+\frac{1}{4}\left[\mathbf{n}_j\cdot(\mathbf{x}-\mathbf{y}_j)\right]D_2(r_j)\right). \tag{7.3b}$$

These sums have the pairwise interaction form (1.1) suitable for a fast summation method; in this case the kernels are tensors, and the particle weights and output velocities are vectors. However the MRS functions (7.1) are somewhat complicated and we know of only two recent applications of fast summation methods to these kernels [33,48] using either the original KIFMM [60] or a later version in which the equivalent densities are defined on coronas (or shells) around each cluster [61]. Note also that when $\epsilon=0$, the MRS kernels reduce to the usual Stokes kernels which are homogeneous (i.e. $k(\alpha\mathbf{x},\alpha\mathbf{y})=\alpha^{\lambda}k(\mathbf{x},\mathbf{y})$ for some constant λ and all $\alpha>0$); some versions of the FMM use this property to improve performance (e.g. [40,51]), but this optimization is not available for the MRS kernels because they are non-homogeneous when $\epsilon\neq 0$. In the next section we demonstrate the capability of the KITC in evaluating the MRS sums (7.2)-(7.3) with $\epsilon\neq 0$.

8 Numerical results

We present results for two examples motivated by recent MRS simulations [48]. In these examples the targets and sources coincide, but this is not an essential restriction. To quantify the accuracy of the KITC we define the relative error,

$$E=\left(\sum_{i=1}^{N}|\mathbf{u}^d(\mathbf{x}_i)-\mathbf{u}^t(\mathbf{x}_i)|^2\Big/\sum_{i=1}^{N}|\mathbf{u}^d(\mathbf{x}_i)|^2\right)^{1/2}, \tag{8.1}$$

where $\mathbf{u}^d(\mathbf{x}_i)$ is the exact velocity computed by direct summation, $\mathbf{u}^t(\mathbf{x}_i)$ is the treecode approximation, and $|\mathbf{u}|$ is the Euclidean norm. In Example 2 where the linear velocity (7.3a) and angular velocity (7.3b) are computed, they are combined into a single vector and the error is computed as in (8.1). All lengths are nondimensional. References to "CPU time" are the total wall-clock run time in seconds.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

The KITC algorithm described above was coded in C++ in double precision and compiled using the Intel compiler icpc with $-$O2 optimization. The source code is available for download (github.com/Treecodes/stokes-treecode). All computations were performed on the University of Wisconsin-Milwaukee Mortimer Faculty Research Cluster; each node is a Dell PowerEdge R430 server with two 12-core Intel Xeon E5-2680 v3 processors at 2.50 GHz and 64 GB RAM; all compute and I/O nodes are linked by Mellanox FDR Infiniband (56Gb/s) and gigabit Ethernet networks. For each example below, results of serial computations using a single core are first presented, followed by results of parallel computations using OpenMP with up to 24 cores on a single node.

8.1 Example 1

The first example simulates microorganisms randomly located in a cube of side length $L=10$, where each microorganism is a pair of particles representing its body and flagella, and the particles exert unit forces in opposite directions along the organism length [48]. The total number of particles is N and we consider five systems with $N=10$K, 80K, 640K, 5.12M, 40.96M. The maximum number of particles in a leaf was set to $N_0=2000$; note however that in this case each cluster has 8 children and the actual leaf size is approximately $N/8^d=1250$, where d is the depth of the tree (some leaves may have a few more or less since the particle locations are random). The microorganism length is $\ell=0.02$ and the MRS parameter is $\epsilon=0.02$. In this example we compute the Stokeslet velocity (7.2).

8.1.1 CPU time versus error

Fig. 2 focuses on intermediate system size $N=640$K, showing the treecode CPU time versus the error E, for MAC parameters in the range $0.4\leq\theta\leq 0.8$ and interpolation degrees $n=1,\cdots,10$ (increasing from right to left). For a given MAC parameter θ, the error E decreases as the degree n increases, but the CPU time increases. The lower envelope of the data gives the most efficient treecode parameters. For example to achieve error $E\leq$ 1e-4, we can choose MAC $\theta=0.7$ and degree $n=7$. Fig. 2 also shows the direct sum CPU time (2249 s) as a horizontal line (the value is independent of the error); the treecode is faster than direct summation for this range of parameters.

8.1.2 Error and CPU time versus system size

Fig. 3 plots the treecode error E (a), and CPU time (b) versus system size N, for MAC parameter $\theta=0.7$ and interpolation degree $n=1,3,5,7,9$. Since it is impractical to carry out the direct sum for the largest system with $N=40.96$M, in that case the error was computed using a sample of 2000 particles and the CPU time was extrapolated from smaller systems. In Fig. 3a for a given degree n, the error first increases with increasing system size N and then it decreases slightly for the two largest systems; nonetheless for a given system size N, the error decreases as the interpolation degree n increases.

Fig. 3b shows that the treecode is faster than direct summation except for the smallest system with $N=10$K (those runs all take less than 1 s). Two reference lines are shown;

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

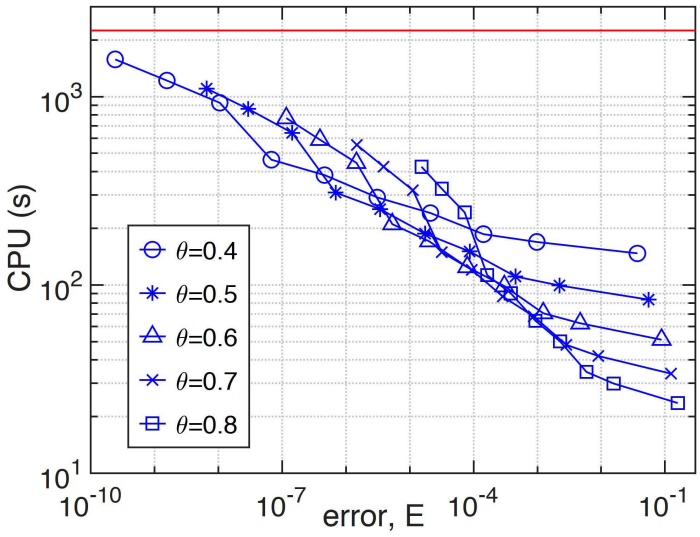

Figure 2: Example 1, regularized Stokeslets in a cube (7.2), MRS parameter $\epsilon=0.02$, system size $N=640$K, direct sum CPU time (2249 s, red horizontal line), treecode CPU time (s) is plotted versus error E (blue symbols), MAC parameter $0.4\leq\theta\leq0.8$, degree $n=1,\cdots,10$ (increasing from right to left).

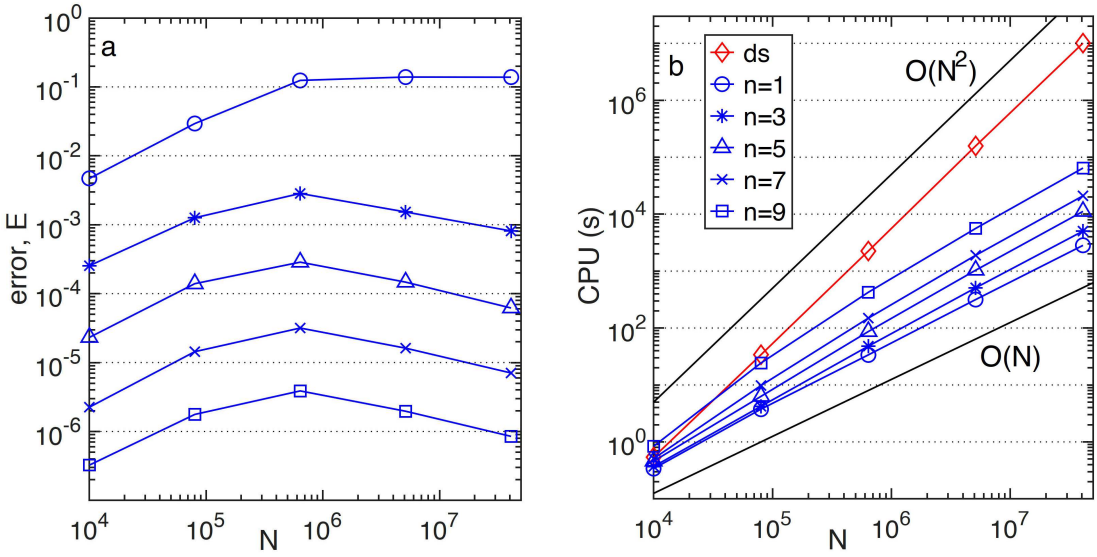

Figure 3: Example 1, regularized Stokeslets in a cube (7.2), MRS parameter $\epsilon=0.02$, (a) treecode error E, (b) CPU time (s) versus system size N (10K, 80K, 640K, 5.12M, 40.96M), treecode MAC parameter $\theta=0.7$, degree $n=1,3,5,7,9$, maximum leaf size $N_0=2000$, direct sum CPU time (ds, red,◊).

the direct sum CPU time is parallel to the $\mathcal{O}(N^2)$ line as expected, while the KITC time is slightly steeper than the $\mathcal{O}(N)$ line. The original Barnes-Hut treecode paper [4] pointed out that the algorithm scales like $\mathcal{O}(N\log N)$; the factor N comes from the loop over target

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

Table 1: Example 1, regularized Stokeslets in a cube (7.2), MRS parameter $\epsilon=0.02$, system size N, CPU time (s) for direct sum and treecode (d,t), speedup (d/t), treecode error (E), results are from Fig. 3 with MAC parameter $\theta=0.7$, degree $n=7$, maximum leaf size $N_0=2000$.

N	direct sum (s), d	treecode (s), t	speedup, d/t	error, E
10K	0.54	0.51	1.06	2.25e-6
80K	34.01	9.78	3.48	1.44e-5
640K	2248.93	149.14	15.08	3.17e-5
5.12M	157,542.13	1892.57	83.24	1.62e-5
40.96M	10,082,696.32	20,966.04	480.91	7.06e-6

particles (line 7 in Algorithm 2) and the factor $\log N$ is the number of levels in the tree; the present results are consistent with this scaling estimate. As expected, for a given degree n the speedup provided by the treecode increases with the system size N. To quantify this observation, Table 1 records the direct sum and treecode CPU time and the treecode error for the five systems using MAC parameter $\theta=0.7$ and degree $n=7$. For the largest system with $N=40.96$M particles, the treecode is more than 480 times faster than direct summation while achieving error $E=7.06$e-6.

8.1.3 Memory consumption

The treecode memory consumption has two main components which are estimated as follows. First, the memory due to the particle coordinates and weights is $\mathcal{O}(N)$, where N is the number of particles in the system; this is the same as in direct summation. Second, the memory due to the modified weights is $\mathcal{O}(n^3N)$, where n is the degree of polynomial interpolation and in this case N represents the number of clusters in the tree (this assumes the number of levels in the tree is $\mathcal{O}(\log N)$). Hence the theoretical memory consumption of the treecode is $\mathcal{O}(n^3N)$, although in practice the $\mathcal{O}(N)$ particle data dominates when the degree n is small.

To see the actual values, Table 2 presents the direct sum and treecode memory consumption (MB) for Example 1; the results were obtained using the Valgrind Massif tool (www.valgrind.org). For the largest system with $N=40.96$M particles, the direct sum memory consumption (3604.80MB) was obtained by extrapolating from the smaller systems. It can be verified that the memory consumption in Table 2 is consistent with the estimates in the previous paragraph; for a given degree n, the memory consumption scales like $\mathcal{O}(N)$, while for a given system size N, beyond the baseline particle data, the additional memory consumption scales like $\mathcal{O}(n^3)$. Over this parameter range, the treecode uses less than 1.65 times as much memory as direct summation, and a KITC computation with $N=40.96$M particles and degree $n=9$ uses about 5.2GB of memory. For comparison, a bbFMM computation with 1M Stokeslets ran out of memory for degree $n>7$ ([19], Fig. 8), although the precise memory consumption value was not given.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

Table 2: Example 1, regularized Stokeslets in a cube (7.2), MRS parameter $\epsilon=0.02$, direct sum and KITC memory consumption (MB), system size N, treecode MAC parameter $\theta=0.7$, degree $n=1,3,5,7,9$, maximum leaf size $N_0=2000$.

system size N	10K	80K	640K	5.12M	40.96M
direct sum memory (MB)	0.88	7.04	56.36	450.60	3604.80
treecode memory (MB)					
$n=1$	0.89	7.09	56.74	453.69	3629.24
$n=3$	0.91	7.27	58.17	465.09	3720.46
$n=5$	0.98	7.74	61.96	495.37	3962.69
$n=7$	1.16	8.66	69.28	553.96	4431.40
$n=9$	1.44	10.16	81.32	650.30	5202.10

8.1.4 Effect of MRS parameter

Table 3 shows the error E for values of the MRS parameter in the range $0.005\leq\epsilon\leq0.08$, with microorganism length $l=\epsilon$ and system size $N=640$K [48]. The KITC used MAC $\theta=0.7$ and degree $n=7$, and the error is on the order of 1e-5. As ϵ increases, the error first increases and then decreases; some error variation with ϵ is expected, but the reason for this particular variation is not clear. It can be noted that error variation with ϵ also occurred in KIFMM simulations (see [48], Table 3).

Table 3: Example 1, regularized Stokelets in a cube (7.2), system size N=640K, MRS parameter ϵ, microorganism length $l=\epsilon$, error E, treecode parameters $\theta=0.7$, $n=7$, maximum leaf size $N_0=2000$.

MRS parameter, ϵ	0.005	0.01	0.02	0.04	0.08
error, E	7.68e-6	2.08e-5	3.17e-5	2.78e-5	1.93e-5

8.1.5 Parallel simulations

We parallelized the direct sum and KITC using OpenMP with up to 24 cores on a single node. In both methods, the loop over target particles was parallelized, and the computation of the modified weights in the KITC was also parallelized.

We consider a system of N=640K regularized Stokeslets; the treecode parameters are $\theta=0.7$, $n=7$, $N_0=2000$, yielding E=3.17e-5. Table 4 shows the CPU time for direct sum and KITC (d,t), parallel speedup and efficiency, and ratio of direct sum and treecode CPU time. Parallelizing the direct sum reduces the CPU time from 2249 s with 1 core to 108 s with 24 cores, yielding parallel efficiency 86%. Parallelizing the KITC reduces the CPU time from 149 s with 1 core to 10 s with 24 cores, yielding parallel efficiency 61%. The KITC is 15 times faster than direct summation with 1 core and 10 times faster with 24 cores.

For comparison, the KIFMM results in [48] employed the Matlab Parallel Computing Toolbox with up to 8 cores. For a system of size N=640K using a 4^3 grid to solve for

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

Table 4: Example 1, regularized Stokeslets in a cube (7.2), parallel performance, system size $N=640$K, MRS parameter $\epsilon=0.02$, KITC parameters $\theta=0.7$, $n=7$, $N_0=2000$, error $E=3.17$e-5, number of cores (nc), CPU time (s) is shown for direct sum and KITC (d,t), parallel speedup $=d_1/d_{nc},t_1/t_{nc}$, parallel efficiency (PE %) $=(d_1/d_{nc})/nc,(t_1/t_{nc})/nc$, parallel KITC speedup (d/t).

nc	d time (s)	d_1/d_{nc}	d PE (%)	t time (s)	t_1/t_{nc}	t PE (%)	d/t
1	2249.71	1.00	100.00	149.75	1.00	100.00	15.02
2	1123.72	2.00	100.00	75.01	2.00	99.82	14.98
4	569.58	3.95	98.74	38.83	3.86	96.41	14.67
8	311.50	7.22	90.28	21.44	6.98	87.31	14.53
12	215.49	10.43	87.00	16.34	9.16	76.37	13.19
24	108.22	20.79	86.62	10.19	14.70	61.23	10.62

the equivalent densities, the error was $E=7.33$e-4, and the CPU time was reduced from 537 s with 1 core to 100 s with 8 cores, yielding parallel efficiency 67% (see [48], Tables 4 and 5). In comparing the KITC results and KIFMM results in [48], the point is not to claim that one method is better than the other; that would require more extensive testing beyond the scope of this work, but we believe the present results indicate that the KITC is a competitive option for fast summation of MRS kernels.

8.2 Example 2

The second example models an array of rods representing cilia or free-swimming flagella [45, 48]. The number of rods is N_r and each rod is a helical curve with M segments, so the total number of particles is $N=N_r(M+1)$. Each rod is parametrized by the z-coordinate and has the form $(x_0+0.3\cos 2z, y_0+0.3\sin 2z, z)$ for $0\le z\le 9$, where the base point (x_0,y_0) lies at a regular Cartesian grid point in the xy-plane and the rods extend in the z-direction. Fig. 4 shows an example with $N_r=16$ rods and $M=25$ segments on each rod, in the domain $[-8,8]\times[-8,8]\times[0,9]$ in $\mathbb{R}^3$.

In this example each particle is a superposition of a regularized Stokeslet and rotlet, and the KITC is applied to compute the linear velocity (7.3a) and angular velocity (7.3b) for a given rod configuration. Starting with $N_r=15^2$ rods with base points in the domain $[-8,8]\times[-8,8]$ as in [48], we increase the number of rods ($N_r=15^2,20^2,30^2,40^2,60^2,80^2$) while expanding the horizontal dimensions of the domain to maintain constant rod density. Each rod has $M=150$ segments, the MRS parameter is $\epsilon=5L/M=0.3$, and each component of the force $\mathbf{f}_j$ and torque $\mathbf{n}_j$ is a random number in $[-1,1]$.

Unlike Example 1 where the particles lie in a cube, in Example 2 the particles lie in a rectangular slab (the z-direction is shorter than the xy-directions). In particular, the slab dimensions vary from $16^2\times 9$ for the smallest system ($N_r=15^2$, $N=33975$) to approximately $85^2\times 9$ for the largest system ($N_r=80^2$, $N=966400$). The tree construction scheme described above yields clusters that are well adapted to the slab geometry of this example.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

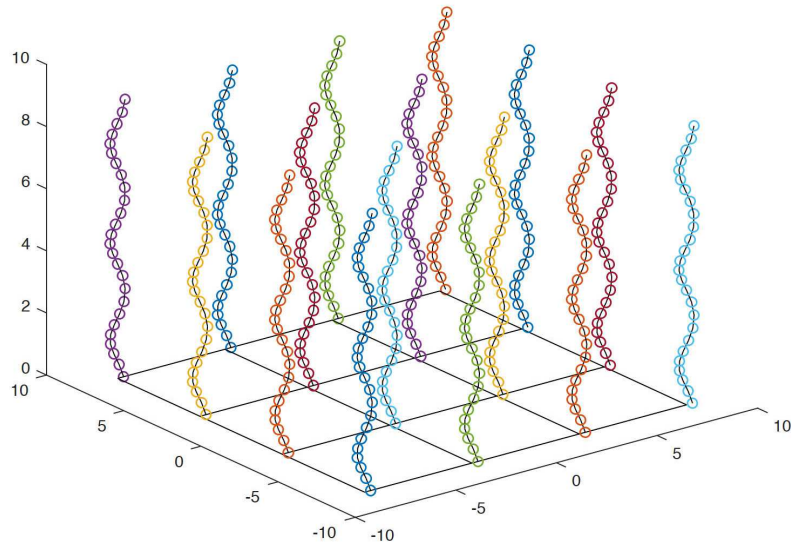

Figure 4: Example 2, regularized Stokeslets and rotlets on an array of helical rods representing cilia or free-swimming flagella [45, 48], $N_r = 16$ rods, $M = 25$ segments on each rod, domain $[-8,8]\times[-8,8]\times[0,9]$ in $\mathbb{R}^3$.

Also note that the regularized Stokeslet/rotlet kernel (7.3) in Example 2 has more terms and is more expensive to evaluate than the regularized Stokeslet kernel (7.2) in Example 1. Hence to better balance the cost of direct particle-particle interactions (4.2) and particle-cluster approximations (4.3), the maximum leaf size was reduced to $N_0 = 1000$ in Example 2.

8.2.1 Error and CPU time versus system size

Fig. 5 presents the treecode error (a) and the direct sum and treecode CPU time (b) versus system size N. The treecode MAC parameter is $\theta = 0.7$ and the degree is $n = 1,3,5,7,9$. The error increases slightly with system size N, but for a given degree n, the error amplitude is comparable to the results in Example 1 for this range of system size. As before, the direct sum CPU time is parallel to the slope 2 line, while the KITC time is slightly steeper than the slope 1 line and is consistent with $\mathcal{O}(N\log N)$ scaling.

8.2.2 Parallel simulations

Table 5 presents results for $N_r = 80^2$ rods, system size $N = 966400$, and with treecode parameters $\theta = 0.7$, $n = 7$, $N_0 = 1000$, yielding error $E = 2.24\text{e-}5$. Parallelizing the direct sum reduces the CPU time from 15651 s with 1 core to 742 s with 24 cores, yielding parallel efficiency 87%, while parallelizing the KITC reduces the CPU time from 720 s with 1 core to 43 s with 24 cores, yielding parallel efficiency 68%. The KITC is 21 times faster than direct summation with 1 core and 17 times faster with 24 cores.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

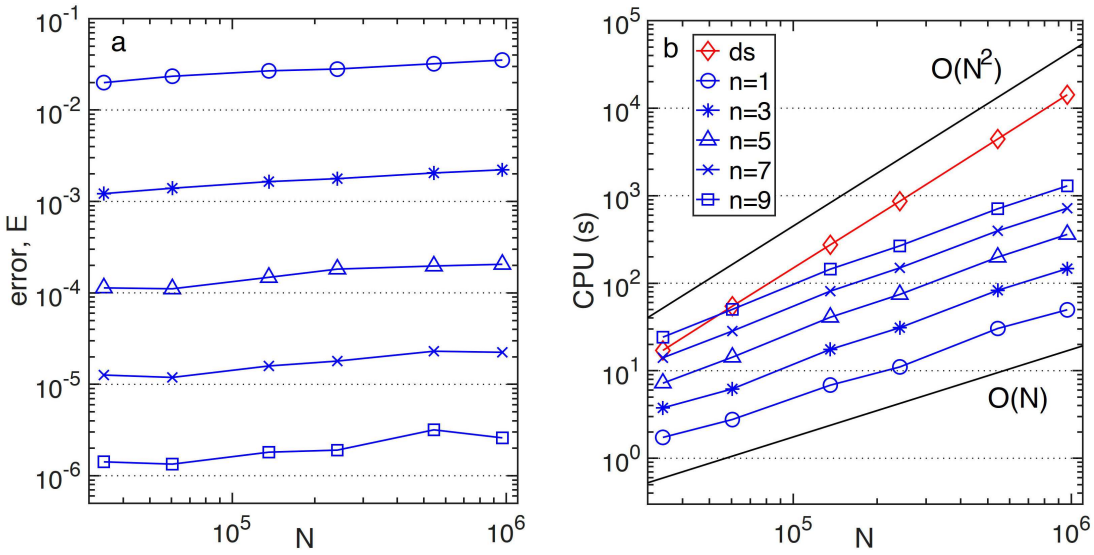

Figure 5: Example 2, regularized Stokeslets and rotlets on an array of helical rods (7.3a)-(7.3b) following [48], (a) KITC error E, (b) CPU time (s) for direct sum (red,$\Diamond$) and KITC (blue, other symbols), number of rods N_r=15^2,20^2,30^2,40^2,60^2,80^2, number of segments per rod M=150, system size $N=N_r(M+1)$, MRS parameter ϵ=0.3, KITC parameters θ=0.7, n=1,3,5,7,9, N_0=1000.

Table 5: Example 2, regularized Stokeslets and rotlets on an array of helical rods (7.3a)-(7.3b) following [48], parallel performance, $N_r=80^2$ rods, system size N=966400, MRS parameter ϵ=0.3, KITC parameters θ=0.7, n=7, N_0=1000, error E=2.24e-5, number of cores (nc), CPU time (s) is shown for direct sum and KITC (d,t), parallel speedup $=d_1/d_{nc}$,t_1/t_{nc}, parallel efficiency (PE %) $=(d_1/d_{nc})/nc$,$(t_1/t_{nc})/nc$, parallel KITC speedup (d/t).

nc	d time (s)	d_1/d_{np}	d PE (%)	t time (s)	t_1/t_{np}	t PE (%)	d/t
1	15651.61	1.00	100.00	720.87	1.00	100.0	21.71
2	7907.10	1.98	98.97	371.91	1.94	96.91	21.26
4	3962.36	3.95	98.75	191.89	3.76	93.91	20.65
8	2120.36	7.38	92.27	110.88	6.50	81.27	19.12
12	1424.27	10.99	91.58	80.35	8.97	74.76	17.73
24	742.85	21.07	87.79	43.63	16.52	68.84	17.03

9 Summary

We presented a kernel-independent treecode (KITC) for fast summation of particle interactions in 3D. The method employs barycentric Lagrange interpolation at Chebyshev points to approximate well-separated particle-cluster interactions. The KITC requires only kernel evaluations, is suitable for non-oscillatory kernels, and relies on the scale-invariance property of barycentric Lagrange interpolation. Numerical results were presented for the non-homogeneous kernels arising in the Method of Regularized Stokeslets

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

(MRS) [48]. For a given level of accuracy, the treecode CPU time scales like $\mathcal{O}(N\log N)$, where N is the number of particles, and a substantial speedup over direct summation is achieved for large systems. The KITC is a relatively simple algorithm with low memory consumption, and this enables a straightforward parallelization; here we employed OpenMP with up to 24 cores on a single node; alternative approaches including distributed memory parallelization for larger systems will be considered in the future [17, 44, 58].

Other kernels for which the KITC may be suitable are the regularized Green's functions used in computing nearly singular integrals (e.g. [5, 54, 55]), the RPY (Rotne-Prager-Yamakawa) tensor for hydrodynamic interactions [37], and the generalized Born potential for implicit solvent modeling [59]. We expect that the KITC can take advantage of several techniques employed in advanced implementations of the KIFMM and bbFMM such as blocking operations, utilizing BLAS routines, AVX and SSE vectorization, and GPU and Phi coprocessing (e.g. [1, 34, 40, 51]). It should be noted that an FMM could be implemented using barycentric Lagrange interpolation for both the target and source variables, and this is also an interesting direction for future study. Finally we mention that the extension to barycentric Hermite interpolation has been carried out for scalar electrostatic kernels [31].

Acknowledgments

We thank the reviewers for careful reading and helpful suggestions to improve the article. We are grateful for a computer allocation on the University of Wisconsin-Milwaukee Mortimer Faculty Research Cluster. We thank Nathan Vaughn and Leighton Wilson for help with programming. Partial support is acknowledged from start-up funds provided by the University of Wisconsin-Milwaukee, NSF grant DMS-1819094, the Michigan Institute for Computational Discovery and Engineering (MICDE), and the Mcubed program at the University of Michigan.

References

[1] E. Agullo, B. Bramas, O. Coulaud, E. Darve, M. Messner and T. Takahashi, Task-based FMM for multicore architectures, SIAM J. Sci. Comput., 36 (2014), C66-C93.
[2] L. af Klinteberg, D. S. Shamshirgar and A.-K. Tornberg, Fast Ewald summation for free-space Stokes potentials, Res. Math. Sci., 4 (2017), Article 1.
[3] C. R. Anderson, An implementation of the Fast Multipole Method without multipoles, SIAM J. Sci. Stat. Comput., 13 (1992), 923-947.
[4] J. E. Barnes and P. Hut, A hierarchical $\mathcal{O}(N\log N)$ force-calculation algorithm, Nature, 324 (1986), 446-449.
[5] J. T. Beale and M.-C. Lai, A method for computing nearly singular integrals, SIAM J. Numer. Anal., 38 (2001), 1902-1925.
[6] J.-P. Berrut and L. N. Trefethen, Barycentric Lagrange interpolation, SIAM Rev., 46 (2004), 501-517.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

[7] S. Börm, L. Grasedyck and W. Hackbusch, Introduction to hierarchical matrices with applications, Eng. Anal. Bound. Elem., 27 (2003), 405-422.
[8] E. L. Bouzarth and M. L. Minion, Modeling slender bodies with the method of regularized Stokeslets, J. Comput. Phys., 230 (2011), 3929-3947.
[9] A. Brandt and A. A. Lubrecht, Multilevel matrix multiplication and fast solution of integral equations, J. Comput. Phys., 90 (1990), 348-370.
[10] H. Cheng, L. Greengard and V. Rokhlin, A fast adaptive multipole algorithm in three dimensions, J. Comput. Phys., 155 (1999), 468-498.
[11] R. Cortez, The method of regularized Stokeslets, SIAM J. Sci. Comput., 23 (2001), 1204-1225.
[12] R. Cortez, L. Fauci and A. Medovikov, The method of regularized Stokeslets in three dimensions: Analysis, validation, and application to helical swimming, Phys. Fluids, 17 (2005), Article 031504.
[13] C. I. Draghicescu and M. Draghicescu, A fast algorithm for vortex blob interactions, J. Comput. Phys., 116 (1995), 69-78.
[14] T. A. Driscoll, N. Hale and L. N. Trefethen, Chebfun Guide, Pafnuty Publications, Oxford, 2014. www.chebfun.org
[15] Z.-H. Duan and R. Krasny, An adaptive treecode for computing nonbonded potential energy in classical molecular systems, J. Comput. Chem., 22 (2001), 184-195.
[16] U. Essmann, L. Perera, M. Berkowitz, T. Darden, H. Lee and L. Pedersen, A smooth particle mesh Ewald method, J. Chem. Phys., 103 (1995), 8577-8593.
[17] H. Feng, A. Barua, S. Li and X. Li, A parallel adaptive treecode algorithm for evolution of elastically stressed solids, Commun. Comput. Phys., 15 (2014), 365-387.
[18] H. Flores, E. Lobaton, S. Méndez-Diez, S. Tlupova, and R. Cortez, A study of bacterial flagellar bundling, Bull. Math. Biol., 67 (2005), 137-168.
[19] W. Fong and E. Darve, The black-box fast multipole method, J. Comput. Phys., 228 (2009), 8712-8725.
[20] W.-H. Geng and R. Krasny, A treecode-accelerated boundary integral Poisson-Boltzmann solver for solvated biomolecules, J. Comput. Phys., 247 (2013), 62-78.
[21] K. Giebermann, Multilevel approximation of boundary integral operators, Computing, 67 (2001), 183-207.
[22] Z. Gimbutas and V. Rokhlin, A generalized Fast Multipole Method for nonoscillatory kernels, SIAM J. Sci. Comput., 24 (202), 796-817.
[23] L. F. Greengard and J. Huang, A new version of the Fast Multipole Method for screened Coulomb interactions in three dimensions, J. Comput. Phys., 180 (2002), 642-658.
[24] L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys., 73 (1987), 325-348.
[25] L. Greengard, The Rapid Evaluation of Potential Fields in Particle Systems, MIT Press, Cambridge, MA 1988.
[26] W. Hackbusch and Z. P. Nowak, On the fast matrix multiplication in the boundary element method by panel clustering, Numer. Math., 54 (1989), 463-491.
[27] D. J. Hardy, M. A. Wolff, J. Xia, K. Schulten and R. D. Skeel, Multilevel summation with B-spline interpolation for pairwise interactions in molecular dynamics simulations, J. Chem. Phys., 144 (2016), Article 114112.
[28] N. J. Higham, The numerical stability of barycentric Lagrange interpolation, IMA J. Numer. Anal., 24 (2004), 547-556.
[29] R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles, Taylor & Francis, Bristol, 1988.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

[30] R. Krasny and L. Wang, Fast evaluation of multiquadric RBF sums by a Cartesian treecode, SIAM J. Sci. Comput., 33 (2011), 2341-2355.
[31] R. Krasny and L. Wang, A treecode based on barycentric Hermite interpolation for electrostatic particle interactions, Comput. Math. Biophys., 7 (2019), 73-84.
[32] R. Kress, Linear Integral Equations, Springer, New York, 2014, third edition.
[33] J. LaGrone, R. Cortez, W. Yan and L. Fauci, Complex dynamics of long, flexible fibers in shear, J. Non-Newton. Fluid Mech., 269 (2019), 73-81.
[34] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin and G. Biros, A massively parallel adaptive fast-multipole method on heterogeneous architectures, Commun. ACM, 55 (2012), 101-109.
[35] P.-D. Létourneau, C. Cecka and E. Darve, Cauchy Fast Multipole Method for general analytic kernels, SIAM J. Sci. Comput., 36 (2014), A396-A426.
[36] P. Li, H. Johnston and R. Krasny, A Cartesian treecode for screened Coulomb interactions, J. Comput. Phys., 228 (2009), 3858-3868.
[37] Z. Liang, Z. Gimbutas, L. Greengard, J. Huang and S. Jiang, A fast multipole method for the Rotne-Prager-Yamakawa tensor and its applications, J. Comput. Phys., 234 (2013), 133-139.
[38] K. Lindsay and R. Krasny, A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow, J. Comput. Phys., 172 (2001), 879-907.
[39] J. Makino, Yet another fast multipole method without multipoles - Pseudoparticle multipole method, J. Comput. Phys., 151 (1999), 910-920.
[40] D. Malhotra and G. Biros, Algorithm 967: A distributed-memory fast multipole method for volume potentials, ACM Trans. Math. Softw., 43 (2016), Article 17.
[41] W. B. March, B. Xiao and G. Biros, ASKIT: Approximate skeletonization kernel-independent treecode in high dimensions, SIAM J. Sci. Comput., 37 (2015), A1089-A1110.
[42] P. G. Martinsson and V. Rokhlin, An accelerated kernel-independent fast multipole method in one dimension, SIAM J. Sci. Comput., 29 (2007), 1160-1178.
[43] W. F. Mascarenhas, The stability of barycentric interpolation at the Chebyshev points of the second kind, Numer. Math., 128 (2014), 265-300.
[44] Y. M. Marzouk and A. F. Ghoniem, *K*-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical *N*-body simulations, J. Comput. Phys., 207 (2005), 493-528.
[45] S. D. Olson, S. Lim and R. Cortez, Modeling the dynamics of an elastic rod with intrinsic curvature and twisting using a regularized Stokes formulation, J. Comput. Phys., 238 (2013), 169-187.
[46] C. Pozrikidis, Boundary Integral and Singularity Methods for Linearized Viscous Flow, Cambridge University Press, Cambridge, UK, 1992.
[47] H.-J. Rack and M. Reimer, The numerical stability of evaluation schemes for polynomials based on the Lagrange interpolation form, BIT, 22 (1982), 101-107.
[48] M. W. Rostami and S. D. Olson, Kernel-independent fast multipole method within the framework of regularized Stokeslets, J. Fluid Struct., 67 (2016), 60-84.
[49] H. E. Salzer, Lagrangian interpolation at the Chebyshev points $x_{n,\nu}=\cos(\nu\pi/n),\nu=0(1)n$; some unnoted advantages, Comput. J., 15 (1972), 156-159.
[50] D. J. Smith, A boundary element regularized Stokeslet method applied to cilia- and flagella-driven flow, Proc. R. Soc. A, 465 (2009), 3605-3626.
[51] T. Takahashi, C. Cecka and E. Darve, Optimization of the parallel black-box fast multipole method on CUDA, in Proceedings of Conference on Innovative Parallel Computing (InPar), 2012, San Jose, CA, USA. DOI: 10.1109/InPar.2012.6339607.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS

[52] J. Tausch, The Fast Multipole Method for arbitrary Green's functions, Contemp. Math., 329 (2003), 307-314.
[53] S. Tlupova and R. Cortez, Boundary integral solutions of coupled Stokes and Darcy flows, J. Comput. Phys., 228 (2009), 158-179.
[54] S. Tlupova and J. T. Beale, Nearly singular integrals in 3D Stokes flow, Commun. Comput. Phys., 14 (2013), 1207-1227.
[55] S. Tlupova and J. T. Beale, Regularized single and double layer integrals in 3D Stokes flow, J. Comput. Phys., 386 (2019), 568-584.
[56] L. N. Trefethen, Approximation Theory and Approximation Practice, SIAM, Philadelphia, 2013.
[57] L. Wang, S. Tlupova and R. Krasny, A treecode algorithm for 3D Stokeslets and stresslets, Adv. Appl. Math. Mech., 11 (2019), 737-756.
[58] M. S. Warren and J. K. Salmon, A parallel hashed oct-tree N-body algorithm, in Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, (1993), 12-21.
[59] Z. Xu, X. Cheng and H. Yang, Treecode-based generalized Born method, J. Chem. Phys., 134 (2011), Article 064107.
[60] L. Ying, G. Biros and D. Zorin, A kernel-independent adaptive fast multipole algorithm in two and three dimensions, J. Comput. Phys., 196 (2004), 591-626.
[61] L. Ying, A kernel independent fast multipole algorithm for radial basis functions, J. Comput. Phys., 213 (2006), 451-457.
[62] B. Zhang, J. Huang, N. P. Pitsianis and X. Sun, A Fourier-series-based kernel-independent fast multipole method, J. Comput. Phys., 230 (2011), 5807-5821.

DOI https://doi.org/10.4208/cicp.OA-2019-0177 | Generated on 2025-04-20 07:59:15
OPEN ACCESS