Commun. Comput. Phys.
doi: 10.4208/cicp.220812.220513a

Vol. **15**, No. 2, pp. 365-387
February 2014

A Parallel Adaptive Treecode Algorithm for Evolution of Elastically Stressed Solids

Hualong Feng[1,2], Amlan Barua[1], Shuwang Li[1] and Xiaofan Li[1,*]

[1] *Department of Applied Mathematics, Illinois Institute of Technology, Chicago, IL 60616, USA.*
[2] *School of Science, Nanjing University of Science and Technology, Nanjing, Jiangsu 210094, P.R. China.*

Received 22 August 2012; Accepted (in revised version) 22 May 2013

Available online 10 September 2013

Abstract. The evolution of precipitates in stressed solids is modeled by coupling a quasi-steady diffusion equation and a linear elasticity equation with dynamic boundary conditions. The governing equations are solved numerically using a boundary integral method (BIM). A critical step in applying BIM is to develop fast algorithms to reduce the arithmetic operation count of matrix-vector multiplications. In this paper, we develop a fast adaptive treecode algorithm for the diffusion and elasticity problems in two dimensions (2D). We present a novel source dividing strategy to parallelize the treecode. Numerical results show that the speedup factor is nearly perfect up to a moderate number of processors. This approach of parallelization can be readily implemented in other treecodes using either uniform or non-uniform point distribution. We demonstrate the effectiveness of the treecode by computing the long-time evolution of a complicated microstructure in elastic media, which would be extremely difficult with a direct summation method due to CPU time constraint. The treecode speeds up computations dramatically while fulfilling the stringent precision requirement dictated by the spectrally accurate BIM.

AMS subject classifications: 65D30, 65N35, 74B05

Key words: Treecode, parallel, elasticity, boundary integral method.

1 Introduction

Crystal growth problem is of primary interest in different fields of science and technology. One example is the production of binary alloys via solid-solid phase transformations. The second/precipitate phase emerges from the mother/matrix phase by lowering

*Corresponding author. *Email addresses:* hfeng8@iit.edu (H. Feng), abarua@hawk.iit.edu (A. Barua), sli@math.iit.edu (S. Li), lix@iit.edu (X. Li)

the temperature and it then grows by diffusion. The morphological evolution of these precipitates raises keen interest in the materials science community, as these microstructures actually control the macroscopic behavior (e.g. mechanical strength) through their interactions (e.g. elasticity) with the surrounding matrix phase.

Computational approaches to this problem include phase field methods and boundary integral methods (BIM) among many others [5, 6, 18, 19, 24, 30, 32]. We will use a boundary integral method to solve the field equations and to track the motion of the interface between the matrix and the precipitate. The main advantage of the BIM is its high accuracy, dimension reduction, and exact treatment of the boundary conditions. Compared with phase field methods, it is not straightforward for BIM to handle phenomena like precipitate merge or splitting. For the problem studied here, we assume topology change does not occur. A review article on boundary integral methods in fluids and materials can be found in [14]. In a boundary integral method, an iterative method (e.g. GMRES) is often used to solve the dense and asymmetric linear systems for the discretized integral equations. In GMRES, to compute the matrix-vector multiplication, a direct summation method requires $\mathcal{O}(N^2)$ operations, where N is the dimension of the linear system or the number of computational points on the interface. The long time computation is prohibitively expensive for precipitates with complicated morphology, as a large N is necessary to resolve the interface. In practice, a fast summation method is used to reduce the computation cost from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$ or $\mathcal{O}(N\log N)$. Examples include the fast multipole method (FMM) [8, 12] and the treecode method [2, 21].

For our proposed problem, Akaiwa and Meiron studied the diffusional effects without elasticity using FMM [1]. Thornton *et al.* [31] performed a computational study of an anisotropic homogeneous problem. In their work, they used the fast multipole method to evaluate the boundary integrals. Jou, Leo, and Lowengrub [15] investigated an isotropic and inhomogeneous problem without fast summation methods. Note that the inhomogeneity requires solution of dipole strength to be used in the boundary integrals. In this paper, we dramatically improve the original methods in [15] by incorporating a time rescaling scheme [3, 20] and a parallel treecode algorithm. The detailed study of the rescaling scheme for this problem has been published in [3]. Here, we focus on the development of the treecode. In particular, we derive recurrence relations in Subsection 3.2 for various kernels used in our integral equations. We also perform the error analysis associated with the treecode approximation. Moreover, using a novel source dividing strategy, we develop an adaptive parallel treecode algorithm in Subsection 3.4. This approach of parallelization can be readily implemented in other treecodes with either uniform or non-uniform point distribution.

The fundamental idea of a treecode is a divide-and-conquer strategy. The Barnes-Hut treecode divides the space evenly into four children in the two-dimensional space [2], and then approximates a cluster of points by a single point at the cluster center with that point carrying all the weights of points in the cluster. Later, this idea is implemented for various kernels [9, 21]. When the size of a cluster is small compared to the distance between a point and the cluster, the treecode approximates point-cluster interactions us-

ing Taylor expansions. Using similar ideas, cluster-cluster interactions through Taylor approximations can also be implemented. In the original form [2], only particle-cluster interactions are employed, resulting in a treecode with time complexity of $\mathcal{O}(N\log N)$. In this paper, we implement point-cluster interactions, derive recurrence relations between the Taylor coefficients of the kernel functions for our problem, and perform the relevant error analysis.

Even with a fast summation method, CPU time constraint can still be an issue for simulating precipitates with complicated morphologies. Among many ways to speed up the treecode, we focus on parallelization because it has the potential to yield significant speedup with a large number of processors. Though we focus on the treecode, we note that several parallel versions of FMM have been developed [11, 16, 25, 35, 36]. One of the pioneering works in treecode parallelization can be found in [33]. Their work is based on spatial decomposition using the Morton order and hashed oct-tree (HOT). Later, a parallel treecode is developed for 3D Ewald summation in molecular dynamics using a modified spatial decomposition method [22]. They showed a linear speedup with at least 64 processors. A more recent progress in treecode parallelization is presented in [34], in which they reported an ideal weak scaling with 16 thousand processors, and an acceptable weak scaling efficiency with 290 thousand processors, if load is balanced a priori.

Most of the previous works present a generic strategy that would work for many different applications of fast summation methods. When implementing parallelization of our treecode, we focus on the boundary integral method applied to the physical problem. We test its performance using the strong scaling law since it is more relevant for our problem. In applications of boundary integral methods, a high accuracy is often required, which in turn demands a high accuracy from the treecode approximation (e.g. the error tolerance $\epsilon \leq 10^{-8}$). Our strategy of parallelization is tailored towards such a treecode with potential to be extended to other parallel treecodes. Specifically, using a novel source diving strategy, we make the programming simple and achieve good speedup for practical purposes. Compared with the traditional target dividing method, our method shows better scalability.

This paper is organized as follows. In Section 2, we describe the physical problem. In Section 3, we develop a treecode. In Section 4, we present numerical results and conclusions.

2 Formulation

2.1 Governing equations

We study microstructural evolutions in 2D numerically. The two-dimensional space is divided into two regions, Ω^M for the matrix phase extending to infinity and Ω^P for the precipitate phase consisting of p disjoint precipitates Ω_i^P, $i=1\cdots,p$. We denote the boundary of the ith precipitate by Γ_i, $i=1,\cdots,p$, and the union of all individual boundaries by

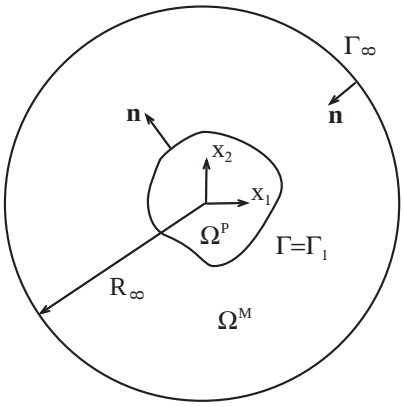


Figure 1: Schematic diagram of a two-phase domain showing one precipitate.

Γ. A schematic showing one precipitate is given in Fig. 1.

Let U be the concentration of the diffusing species in the matrix phase. Under a quasi-static assumption, U satisfies the Laplace's equation $\nabla^2 U=0$ in $\Omega^M \subset R^2$ subject to $U=\kappa+ZG_{\rm el}$ on $\Gamma=\cup_{i=1}^n \Gamma_i$, where κ is the mean curvature, $G_{\rm el}$ is the elastic energy density [15], and Z characterizes the relative contribution of elasticity and surface energies. The far-field condition is taken to be

$$\lim_{R_\infty\to\infty}\int_{\Gamma_\infty}(-\nabla U\cdot n)\,{\rm d}s=2\pi J, \tag{2.1}$$

where s is arclength and $2\pi J$ is mass flux. Once the concentration U is obtained by solving the Laplace equation, the velocity of the interface can be computed by taking normal derivative of the concentration, $V_i=\frac{\partial U}{\partial n}$ on Γ_i.

Note that to solve the diffusion problem, the boundary condition requires a solution to the two phase elasticity problem for the domain Ω^M and Ω^P. Assuming there is no body force, the elasticity problem is given by $\sigma^\chi_{ij,j}=0$ in Ω^χ, $u_i^P=u_i^M$ on Γ, $\sigma_{ij}^P\widehat{n}_j=\sigma_{ij}^M\widehat{n}_j$ on Γ, subject to the boundary condition at infinity $\lim_{r\to\infty}\epsilon_{ij}^M=\epsilon_{ij}^0$, where ϵ_{ij} and σ_{ij} are the strains and stresses, and where χ can be either matrix or precipitate [15].

After the elasticity equations are solved, we compute the elastic energy density

$$G_{\rm el}=\frac{1}{2}\sigma_{ij}^P(\epsilon_{ij}^P-\epsilon_{ij}^T)-\frac{1}{2}\sigma_{ij}^M\epsilon_{ij}^M+\sigma_{ij}^M(\epsilon_{ij}^M-\epsilon_{ij}^P), \tag{2.2}$$

which appears in the boundary condition for the diffusion equation. In the formula for $G_{\rm el}$, ϵ_{ij}^T is the misfit strain of the precipitate. In this work, we focus our study on the applied strain.

2.2 Discretizations

In this subsection, we describe a boundary integral method to simulate the dynamics of the precipitates. Following [15], we use the complex variable notation $z=x_1+ix_2$ to

denote points on the interface. The fundamental solutions for the elasticity equations are given by the Kelvin solution in terms of the displacement and traction,

$$U_{jk}(z,z') = \frac{1}{8\pi\mu(1-\nu)}\left[(3-4\nu)\ln\left(\frac{1}{r}\right)\delta_{jk} + r_{,j'}r_{,k'}\right], \tag{2.3}$$

$$T_{jk}(z,z') = \frac{-1}{4\pi(1-\nu)r}\left[\frac{\partial r}{\partial n'}\left((1-2\nu)\delta_{jk} + 2r_{,j'}r_{,k'}\right) + (1-2\nu)(n'_j r_{,k'} - n'_k r_{,j'})\right], \tag{2.4}$$

where $r = |z'-z|$, $r_{,k'} = \frac{\partial r}{\partial x'_k}$ and n'_k is the kth component of the normal at point z'.

Using the fundamental solutions, we write the solution to the elasticity problem as a set of boundary integral equations. For the ith precipitate, it reads

$$\frac{1}{2}u_j^P + \int_{\Gamma_i} u_k^P T_{jk}^P \mathrm{d}s' - \int_{\Gamma_i} t_k^P U_{jk}^P \mathrm{d}s' = \int_{\Gamma_i} t_k^T U_{jk}^P \mathrm{d}s', \quad j=1,2, \tag{2.5}$$

where u_j^P is the displacement vector and t_j^P is the traction vector. The quantity $t_j^T = \sigma_{jk}^T n_k$ on the right hand side is a given traction due to the misfit strain [15].

Similarly, the boundary integral equation (with u_j^M and t_j^M as unknowns) for the matrix phase is

$$\frac{1}{2}u_j^M - \int_{\Gamma} u_k^M T_{jk}^M \mathrm{d}s' + \int_{\Gamma} t_k^M U_{jk}^M \mathrm{d}s' = \frac{1}{2}u_j^0 - \int_{\Gamma} u_k^0 T_{jk}^M \mathrm{d}s' + \int_{\Gamma} t_k^0 U_{jk}^M \mathrm{d}s', \tag{2.6}$$

where u_k^0 is the kth component of the displacement due to an applied stress and t_k^0 is the corresponding traction. Note that in Eq. (2.6), the integration is over the entire interface, while the integration in Eq. (2.5) is only over the interface Γ_i.

Since the traction and the displacement are continuous across the interface, $u_j^P = u_j^M$ and $t_j^P = t_j^M$ on Γ, we drop the superscripts and simply write these quantities as u_j and t_j respectively. Then Eqs. (2.5) and (2.6) become

$$\frac{1}{2}u_j + \int_{\Gamma_i} u_k T_{jk}^P \mathrm{d}s' - \int_{\Gamma_i} t_k U_{jk}^P \mathrm{d}s' = \int_{\Gamma_i} t_k^T U_{jk} \mathrm{d}s', \tag{2.7}$$

$$\frac{1}{2}u_j - \int_{\Gamma} u_k T_{jk}^M \mathrm{d}s' + \int_{\Gamma} t_k U_{jk}^M \mathrm{d}s' = \frac{1}{2}u_j^0 - \int_{\Gamma} u_k^0 T_{jk}^M \mathrm{d}s' + \int_{\Gamma} t_k^0 U_{jk}^M \mathrm{d}s'. \tag{2.8}$$

These $2p+2$ integrals equations, where p is the number precipitates, must be solved to obtain u_1, u_2, t_1, and t_2. To solve these equations numerically, we discretize the integrals using a spectrally accurate alternating point quadrature [15,28], and use a preconditioned GMRES method [15,26] for solving the discretized equations.

The boundary integral equation for the exterior diffusion problem, in terms of an unknown dipole density potential function ϕ on Γ and unknown p source terms $A_1, A_2, \cdots, A_p$,

is given by the following set of equations [10,23]

$$\left(-\frac{1}{2}\mathcal{I}+K\right)[\phi]+\sum_{i=1}^{p}A_i\log|z(s)-S_i|=\kappa+ZG_{\rm el}, \tag{2.9}$$

$$\sum_{i=1}^{p}A_i=J, \tag{2.10}$$

$$\int_{\Gamma_i(t)}\phi(s')\,\mathrm{d}s'=0,\quad i=1,\cdots,p-1, \tag{2.11}$$

where $S_i=x_{1i}^s+ix_{2i}^s$ is a point inside the closed interface Γ_i. $\mathcal{I}$ is the identity operator and K is the integral operator defined by

$$K[\phi](s)=\frac{1}{2\pi}\int_{\Gamma(t)}\phi(s')\left[\frac{\partial}{\partial n(s')}\log\left|z(s')-z(s)\right|+1\right]\mathrm{d}s'. \tag{2.12}$$

We discretize Eqs. (2.9) and (2.11) in a similar fashion to the elasticity problem, and solve the discretized system for ϕ and A_k with a preconditioned GMRES method. Once the diffusion problem is solved, we compute the normal velocity V_i of the interface Γ_i using the Dirichlet-Neumann map,

$$\begin{aligned}V_i(s)=&\frac{1}{2\pi}\int_{\Gamma}\phi_{,s'}\frac{\partial}{\partial s}\log\left|z(s')-z(s)\right|\mathrm{d}s'\\&+\sum_{k=1}^{p}A_k\frac{(x_1(s)-x_{1k}^s)x_{2,s}-(x_2(s)-x_{2k}^s)x_{1,s}}{(x_1(s)-x_{1k}^s)^2+(x_2(s)-x_{2k}^s)^2},\end{aligned} \tag{2.13}$$

where $x_{j,s}=\frac{\partial x_j(s)}{\partial s}$, $j=1,2$.

To make our presentation of the parallel treecode more precise, we describe the numerical integration of an integral in Eq. (2.7),

$$\int_{\Gamma_i}u_1T_{11}^p\,\mathrm{d}s'. \tag{2.14}$$

Using an alternating point quadrature, the discretization of the integral (2.14) yields the two sums at the computational point z_l,

$$\int_{\Gamma_1}u_1T_{11}^p\,\mathrm{d}s'\approx\begin{cases}\sum\limits_{m=1}^{M}T_{11}^p(z_l,z_{2m-1})w_{2m-1}, & \text{if } l \text{ is even},\\ \sum\limits_{m=1}^{M}T_{11}^p(z_l,z_{2m})w_{2m}, & \text{if } l \text{ is odd},\end{cases} \tag{2.15}$$

where w_j is the weight $u_1(z_j)\,\mathrm{d}s'$ and we assume there are $N=2M$ computational points on the interface Γ_i. In the GMRES solver, these two sums are evaluated with a fast adaptive treecode.

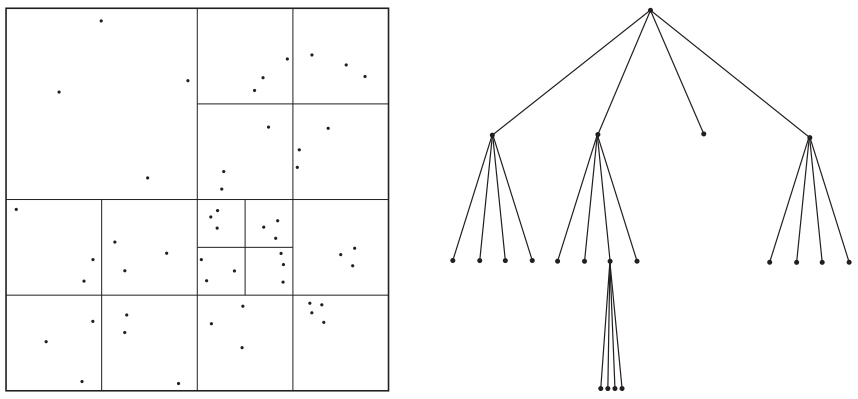
3 The treecode and its parallelization

In this section, we present a treecode for evaluating the boundary integrals in Eqs. (2.7), (2.8), and (2.12). In Subsection 3.1, we present an outline of the treecode. In Subsection 3.2, we derive recurrence relations between the Taylor coefficients for all kernel functions in this problem. We then perform an error analysis of the Taylor approximations. In Subsection 3.3, we provide a description of the algorithm. In Subsection 3.4, we show a new way of parallelizing the algorithm.

3.1 Outline

In a treecode, the space of computational points is first divided hierarchically to form a tree. In general, the points making contributions (called source points) and the points receiving the contributions (called target points) do not have to be the same set of points. Fig. 2 shows a schematic of a tree structure with 4 levels in 2D, where each node represents a cluster of points. In this work, we use node and cluster interchangeably. A parent node contains all the points in its child nodes. When the distance between a target point and a cluster of source points is large relative to the radius of the cluster, the interaction between the target and the cluster is evaluated with a Taylor approximation. We write $\mathbf{x}=(x_1,x_2)$, or in complex variable notation $z=x_1+ix_2$. Let $c=\{\mathbf{x}_i,i=1,\cdots,N_c\}$ be a cluster of N_c source points centered at $\mathbf{x}_c$ and w_i be their corresponding weights. For a kernel function $\Phi(\mathbf{x})$, the interaction between a distant target point $\mathbf{x}$ and the cluster c of source points $\mathbf{x}_i$ is

$$\sum_{i=1}^{N_c}\Phi(\mathbf{x}-\mathbf{x}_i)w_i. \tag{3.1}$$



(a) (b)

Figure 2: A schematic of the physical tree and the logic tree with depth 4. The tree is formed adaptively to reflect non-uniform distribution of computational points, and each node in the logic tree (b) corresponds to a rectangle in the physical tree (a).

Expanding the kernel in a Taylor series about $\mathbf{x}-\mathbf{x}_c$, we obtain

$$\begin{aligned}\sum_{i=1}^{N_c}\Phi(\mathbf{x}-\mathbf{x}_i)w_i &= \sum_{i=1}^{N_c}\Phi(\mathbf{x}-\mathbf{x}_c+\mathbf{x}_c-\mathbf{x}_i)w_i \\ &= \sum_{i=1}^{N_c}\sum_{\mathbf{k}}\frac{1}{\mathbf{k}!}D^{\mathbf{k}}\Phi(\mathbf{x}-\mathbf{x}_c)(\mathbf{x}_c-\mathbf{x}_i)^{\mathbf{k}}w_i \\ &= \sum_{\mathbf{k}}b_{\mathbf{k}}(\mathbf{x}-\mathbf{x}_c)m_{\mathbf{k}}(c),\end{aligned} \tag{3.2}$$

where $\mathbf{k}=(k_1,k_2)$ is the 2D multi-index, $D^{\mathbf{k}}$ is the $\mathbf{k}$-th differentiation operator,

$$b_{\mathbf{k}}(\mathbf{x}-\mathbf{x}_c)=\frac{1}{\mathbf{k}!}D^{\mathbf{k}}\Phi(\mathbf{x}-\mathbf{x}_c) \tag{3.3}$$

is the $\mathbf{k}$-th Taylor coefficient, and

$$m_{\mathbf{k}}(c)=\sum_{i=1}^{N_c}(\mathbf{x}_c-\mathbf{x}_i)^{\mathbf{k}}w_i \tag{3.4}$$

is the $\mathbf{k}$th moment of the cluster c about the center $\mathbf{x}_c$. We retain the terms of order $|\mathbf{k}|<q$ in the infinite series in Eq. (3.2) and get a q-th order point-cluster approximation

$$\sum_{i=1}^{N_c}\Phi(\mathbf{x}-\mathbf{x}_i)w_i\approx\sum_{|\mathbf{k}|<q}b_{\mathbf{k}}(\mathbf{x}-\mathbf{x}_c)m_{\mathbf{k}}(c). \tag{3.5}$$

We show in Subsection 3.2 that the error in the approximation Eq. (3.5) is $\mathcal{O}(h^q)$, where h shown in Fig. 3 is the ratio of the cluster radius r to the distance R between the target point and the cluster center,

$$h=\frac{r}{R}=\frac{\max\{|\mathbf{x}_c-\mathbf{x}_i|,i=1,\cdots,N_c\}}{|\mathbf{x}-\mathbf{x}_c|}. \tag{3.6}$$

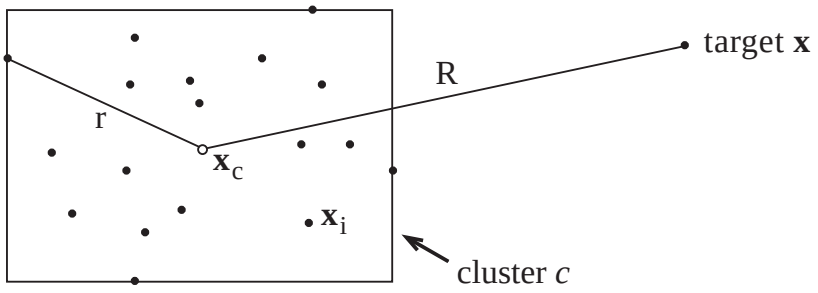


Figure 3: Interaction between a point and a distant cluster is evaluated with a q-th order Taylor approximation when $h=r/R$ is small, and the error behaves like $\mathcal{O}(h^q)$ as $q\to\infty$.

Cluster moments $m_{\mathbf{k}}(c)$ in Eq. (3.4) are independent of the target point $\mathbf{x}$, and the Taylor coefficients $b_{\mathbf{k}}(\mathbf{x}-\mathbf{x}_c)$ in Eq. (3.3) are independent of the source points $\mathbf{x}_i$ in the cluster. Once a cluster moment is computed, it is stored and used repeatedly for computing other target point-cluster interactions. These are the main ideas which a treecode uses to reduce the cost of the N-body problem.

3.2 Recurrence relations for Taylor coefficients and error analysis

When evaluating the point-cluster approximation Eq. (3.5), one needs the Taylor coefficients $\{b_{\mathbf{k}}\}$ of the kernel functions. We next derive a recurrence relation such that the higher-order Taylor coefficients can be computed from the lower-order ones.

All the kernel functions in Eqs. (2.3), (2.4), and (2.12) for the elasticity and diffusion problems are combinations of the following functions

$$\log r, \quad \frac{x_i}{r^2}, \quad \frac{x_i x_j}{r^2}, \quad \frac{x_i}{r^2}\frac{x_i x_j}{r^2}, \quad i,j=1,2, \tag{3.7}$$

where $r^2=x_1^2+x_2^2$. Note that the $\log r$ kernel can be converted to be a combination of the latter three kernels in Eq. (3.7) through integration by parts [15].

If $b_{\mathbf{k}}(\mathbf{x})$ is the $\mathbf{k}$th Taylor coefficient of a kernel function $\Phi(\mathbf{x})$, then the $\mathbf{k}$th Taylor coefficients $\mathbf{a}_{\mathbf{k}}(\mathbf{x})$ of its gradient $\nabla\Phi(\mathbf{x})$ are

$$\mathbf{a}_{\mathbf{k}}(\mathbf{x})=\sum_{i=1}^{2}(k_i+1)b_{\mathbf{k}+\mathbf{e}_i}\mathbf{e}_i, \tag{3.8}$$

where $\mathbf{e}_i$ is the ith Cartesian basis vector. We use this relation to simplify computations of Taylor coefficients. One can verify that all kernels in expression (3.7) are related to the following two kernel functions

$$\phi=\log r \quad \text{and} \quad \psi=\frac{x_1^2}{r^2} \tag{3.9}$$

as follows

$$\log r=\phi, \tag{3.10}$$

$$\left(\frac{x_1}{r^2},\frac{x_2}{r^2}\right)=\nabla\phi, \tag{3.11}$$

$$\left(\frac{x_1^2}{r^2},\partial_{x_1}\frac{x_1x_2}{r^2},\frac{x_2^2}{r^2}\right)=(\psi,\partial_{x_2}(\phi+\psi),1-\psi), \tag{3.12}$$

$$\left(\frac{x_1^3}{r^4},\frac{x_1^2x_2}{r^4},\frac{x_1x_2^2}{r^4},\frac{x_2^3}{r^4}\right)=(\partial_{x_1}(\phi-\psi/2),\partial_{x_2}(-\psi/2),\partial_{x_1}(\psi/2),\partial_{x_2}(\phi+\psi/2)). \tag{3.13}$$

We note that in Eq. (3.12) the second component reads $\partial_{x_1}\frac{x_1x_2}{r^2}=\partial_{x_2}(\phi+\psi)$, meaning that we use Eq. (3.8) to get Taylor coefficients of the function $\partial_{x_1}\frac{x_1x_2}{r^2}$ from those of $(\phi+\psi)$, and

then use Eq. (3.8) backwards to get the Taylor coefficients of the function $\frac{x_1x_2}{r^2}$. With the help of Eqs. (3.10)-(3.13) and (3.8), we derive the recurrence relations between the Taylor coefficients for ϕ and ψ.

The kernel $\phi=\log r$ satisfies the equation

$$r^2D_{x_1}\phi-x_1=0. \tag{3.14}$$

We apply $D_{x_1}^{k_1-1}$ to Eq. (3.14) and use the product rule of differentiation to obtain

$$r^2D_{x_1}^{k_1}\phi+2(k_1-1)x_1D_{x_1}^{k_1-1}\phi+(k_1-1)(k_1-2)D_{x_1}^{k_1-2}\phi=0. \tag{3.15}$$

Applying $D_{x_2}^{k_2}$ to the above equation and using the definition of $b_{\mathbf{k}}$ we get

$$k_1r^2b_{\mathbf{k}}+2k_1\sum_{i=1}^{2}x_ib_{\mathbf{k}-\mathbf{e}_i}-2x_1b_{\mathbf{k}-\mathbf{e}_1}+k_1\sum_{i=1}^{2}b_{\mathbf{k}-2\mathbf{e}_i}-2b_{\mathbf{k}-2\mathbf{e}_1}=0. \tag{3.16}$$

This is a recurrence relation for computing the Taylor coefficients of the kernel $\phi=\log r$. The relation involves three consecutive orders of derivatives. If $k_i<0$ for either i in $\mathbf{k}=(k_1,k_2)$, let $b_{\mathbf{k}}=0$. Similarly, we derive a recurrence relation for the Taylor coefficients $b_{\mathbf{k}}$ of the kernel $\psi=\frac{x_1^2}{r^2}$,

$$r^2b_{\mathbf{k}}+2\sum_{i=1}^{2}x_ib_{\mathbf{k}-\mathbf{e}_i}+\sum_{i=1}^{2}b_{\mathbf{k}-2\mathbf{e}_i}=0. \tag{3.17}$$

We note that since all kernel functions in Eq. (3.7) are either harmonic or bi-harmonic functions, it is possible to derive a fast multipole method. While a q-th order Taylor expansion is of complexity $\mathcal{O}(q^2)$, a q-th order multipole expansion is only $\mathcal{O}(q)$. In this sense, an FMM reduces the dimension of the expansions from two to one, and this effect will be studied in a future work.

We next analyze the error in the q-th order Taylor approximation (3.5) and rewrite,

$$\sum_{i=1}^{N_c}\Phi(\mathbf{x}-\mathbf{x}_i)w_i\approx\sum_{|\mathbf{k}|<q}b_{\mathbf{k}}(\mathbf{x}-\mathbf{x}_c)m_{\mathbf{k}}(c). \tag{3.18}$$

When the kernel function Φ is $\log r$, it is shown in [12] that the q-th order Taylor approximation (3.18) incurs an error of $\mathcal{O}(h^q)$, where $h=r/R$ is defined in Eq. (3.6). While it is possible to perform error analysis of the Taylor approximation (3.18) when the kernel function Φ is $\psi=\frac{x_1^2}{r^2}$ by studying the recurrence relation (3.16), it would be cumbersome. Instead, following [12], we use complex analysis [7] to perform an error estimate of the Taylor approximation (3.18) for $\psi=\frac{x_1^2}{r^2}$.

Being a bi-harmonic function, $\psi=\frac{x_1^2}{r^2}$ can be written as the real or imaginary part of $\tilde{z}f(z)+g(z)$, where both $f(z)$ and $g(z)$ are complex analytic functions. Letting $z=x_1+ix_2$, we write

$$\psi=\frac{x_1^2}{r^2}=\frac{1}{2}\mathrm{Re}\Big(1+\frac{\tilde{z}}{z}\Big).$$

The formal power series

$$\psi(z-z_i)=\frac{1}{2}\mathrm{Re}\left(1+\frac{\tilde{z}-\tilde{z_i}}{z-z_i}\right)=\frac{1}{2}+\mathrm{Re}\left(\frac{\tilde{z}}{z}-\frac{\tilde{z_i}}{z}\right)\sum_{k=0}^{\infty}\frac{z_i^k}{z^k}$$

converges if $|z_i|<|z|$. We truncate this series to obtain

$$\psi(z-z_i)\approx\frac{1}{2}+\mathrm{Re}\left(\frac{\tilde{z}}{z}-\frac{\tilde{z_i}}{z}\right)\sum_{k<q}\frac{z_i^k}{z^k}, \tag{3.19}$$

and clearly the error incurred by this truncation is $\mathcal{O}\left(\left|\frac{z_i}{z}\right|^q\right)$. One can check that the approximation in Eq. (3.19) is the same as that in Eq. (3.18) when the kernel function Φ is $\psi=\frac{x^2}{r^2}$, assuming the weight w_i in Eq. (3.18) to be 1 and the cluster center $\mathbf{x}_c$ to be the origin. We conclude that error in the Taylor approximation (3.18) for the cluster c is

$$\mathcal{O}(h^q|m_0(c)|), \tag{3.20}$$

where the zeroth moment $m_0(c)$ is simply the total weight of the points in c.

3.3 Description of algorithm

The treecode forms the tree by dividing the space of computational points into rectangular panels (clusters) successively, and each panel stores its x and y boundaries. The division process is adaptive in three aspects. First, it may divide a parent cluster into either two or four child clusters. If the aspect ratio is larger than $\sqrt{2}$, it only divides the parent cluster into two child clusters with division happening at the middle in the longer direction. Otherwise, four child clusters are formed by dividing at the center in both directions. The second aspect of the adaptivity is that the subdivision of space stops when a cluster contains a number points less than a preset threshold N_0. It has been observed that a full subdivision of the space of computational points, which results in a tree with more levels, may slow down the algorithm. Third, once a cluster is divided, the child clusters (panels) are shrunk to the smallest rectangle that contains all the points. Shrinking to the smallest rectangle improves the accuracy of error estimates. After the panel is shrunk, the cluster center and the total weights of points in the cluster are computed and stored. The center will be used to evaluate the moments of the points within the cluster, and the weights will be used to estimate the error in the Taylor approximation (3.5).

With a user-prescribed error tolerance ϵ, each target point $\mathbf{x}$ interacts with the tree, starting from the root cluster. The following is the recursive divide-and-conquer process. For a certain cluster c, the algorithm uses the error criterion to check whether the interaction between $\mathbf{x}$ and c can be evaluated with a q-th order Taylor approximation. If yes, the algorithm compares the CPU time needed for a q-th order approximation method and a direction summation method, and then chooses the faster one. Stand-alone tests are performed to estimate the CPU time for a q-th order Taylor approximation, and the CPU

time for direct summations. Before a q-th order Taylor approximation can be done, the algorithm may need to compute the q-th moment of c if it is not available, and flag it as available so it may be used by other target points. If no, there may be two cases. First, the cluster c has *chd_cnt* child clusters, then $\mathbf{x}$ interacts with each child cluster but uses ϵ/chd_cnt as the error tolerance. Second, c is a leaf and hence does not have child clusters, then $\mathbf{x}$ interacts with c through direct summations.

Algorithm 3.1: The treecode

```
program main
construct tree
for i=1:N do
   compute_interaction(x_i, root, ε)
end for

subroutine compute_interaction(x,c,ε)
if particle-cluster interaction b/w x and cluster c incurs error (r/R)^q < ε then
   use q-th order Taylor approximation
else
   if c is a leaf then
      use direct summation
   else
      for each child of c (# children of c: chd_cnt) do
         call compute_interaction(x, child, ε/chd_cnt)
      end for
   end if
end if
```

To summarize, several tuning parameters are used to control the accuracy and the efficiency of the treecode. First, when the root cluster is divided into hierarchical levels of clusters, it may not be optimal to reach the lowest level where every leaf only has one point. Instead, the treecode sets a parameter N_0 so that any cluster containing less than N_0 points is not divided further. Second, the treecode uses varying order Taylor approximations. Computing and storing a large number of Taylor coefficients can add an overhead slowing the algorithm. A maximum allowable order of expansion $Q_{\max}$ is set in the treecode. Whether an expansion of order $Q \leq Q_{\max}$ or a direct summation should be used depends on which is faster. Third, a rigorous error control is available for the treecode. The code takes a tolerance ϵ as an input parameter such that the absolute error is bounded by ϵ.

3.4 Parallelization of the treecode

In this subsection, we describe treecode parallelization. We focus on Message Passing Interface (MPI) and do not discuss OpenMP or GPU or any hybrid implementations. In the original Barnes-Hut treecode [2], only first order Taylor approximations are used for

point-cluster interactions. For such a treecode, the force evaluation part (with the original astrophysics applications in mind) takes over 90% of the total CPU time [29]. The remaining CPU time is devoted to the other parts (e.g. constructing the tree, computing center-of-mass, updating point locations for each time step, etc.). Parallelization of the Barnes-Hut treecode is relatively simple in that only the force evaluation part needs to be parallelized since it consumes most of the CPU time [29]. However, the Barnes-Hut treecode is inadequate in our study since accuracies in the order of 10^{-8} or higher are required for our long-time simulations. Therefore, a treecode employing higher order Taylor approximations is necessary. For such a treecode, moment computation takes a significant portion of the CPU time. To implement parallelization with good scalability, both moment computation and force evaluation need to be parallelized, which is challenging since the two parts have to be parallelized in a coherent way.

A natural strategy to parallelize a treecode is to divide the target points among the processors. Each processor forms a tree with all source points either alone or through collaboration between processors. Then the processor computes contributions from these source points on the target points assigned to it. Once this evaluation is complete, the processor communicates the results to other processors through broadcast or collective communication. We call this the *target dividing strategy*. To optimize this parallelization, data locality is desired. Namely, all the target points assigned to a processor should be as clustered as possible. This is because two target points nearby will see similar traverse orders through the tree during the divide-and-conquer process in the treecode. Hence, when the treecode works on two target points nearby, it is likely to use the same sets of moments. Thus, a processor needs less amount of information from other processors if target points assigned to it are more clustered. However, this conceptually simple strategy is not straightforward to implement, because large amount of data communication between processors will occur. To the best of our knowledge, this is the strategy used in previous parallelizations of the treecode.

Instead of the target dividing strategy, we propose a contra-intuitive method in which the source points are divided among the processors. Each processor forms its own tree out of the source points assigned to it, and computes contributions from these source points on all the target points. Once this evaluation is complete, the processor communicates the results to the head processor. The head processor adds up the contributions on each target from all sources, and broadcasts the results to other processors. We call this the *source dividing strategy*. Similar to the target dividing strategy, data locality is also desired for the source dividing strategy. This is because with a panel consisting of points that are more clustered, Taylor approximations of the point-cluster interaction between a target point and this cluster converge faster. With this new approach, there is no communication between processors when each processor forms its own tree, and then computes interactions between the tree and all the target points. This strategy has its own shortage as discussed in Section 4, but the downside is overweighed by the advantage it brings.

In Fig. 4, we illustrate workload division using sixteen processors. We assume the general setting that the target points and the source points are different sets, which is con-

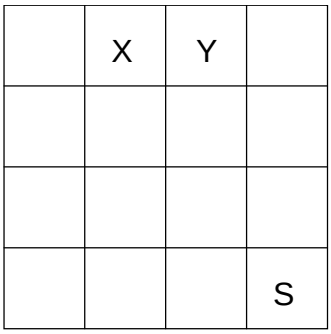
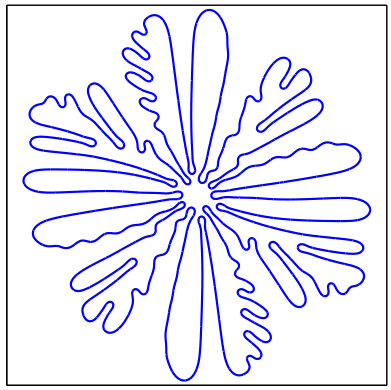


Figure 4: In the target dividing strategy, each processor is assigned the target points in a subregion (using sixteen processors for illustration). While in the source dividing strategy, each processor is assigned the source points in a subregion.

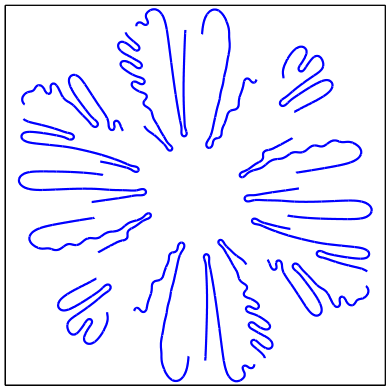
sistent with the alternating point quadrature rule used in this paper. In a target dividing strategy, target points in region X and Y are assigned to processors $\mathcal{X}$ and $\mathcal{Y}$ respectively. For each target point, the corresponding processor needs to compute contributions from all source points. In this process, it often occurs that a target point x in X and a target y in Y need to do point-cluster interactions with source points in region S. To do this, moments of sources in S about its center are needed and shared between processors $\mathcal{X}$ and $\mathcal{Y}$. Therefore, frequent communications are required for the sharing. Since computing moments takes a significant amount of the CPU time, for optimal performance the processors should get a moment from another processor if it is available there rather than compute the moment themselves. In our source dividing strategy, the source points in region S are assigned to a processor, say $\mathcal{S}$. Processor $\mathcal{S}$ forms a tree consisting of all source points in S, then it computes the contribution from these source points on all target points. Communications are minimized, since they only occur at the end when all processors have finished computing contributions from sources points assigned to them. In Section 4, we show that our source dividing strategy is more efficient than the target dividing strategy.

We now discuss data locality and leave the discussion of load balancing to Section 4. We will examine the speed-up factor of the parallel treecode with both randomly generated data and real physical data. For random data, since they are uniformly generated in a square, it is simple to preserve data locality in point division. If P is the number of processors, one divides the square evenly into P subregions with best aspect ratio possible, and assigns the points in each subregion to one processor.

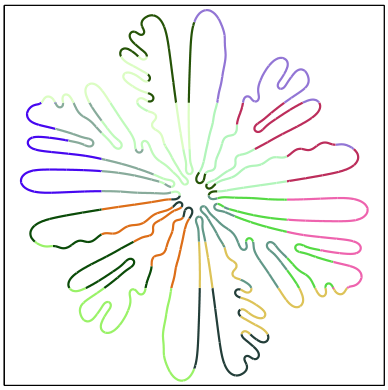
For real physical data, preserving data locality becomes challenging. For example, we take the points on the interface shown in Fig. 5(a), which is reproduced from Fig. 7(f). These computational points are not distributed in a rectangular region, but are confined within a rounded region. We present three ways to partition them and discuss the property of preserving data locality for each. We assume the number of processors is sixteen in the parallel computation.



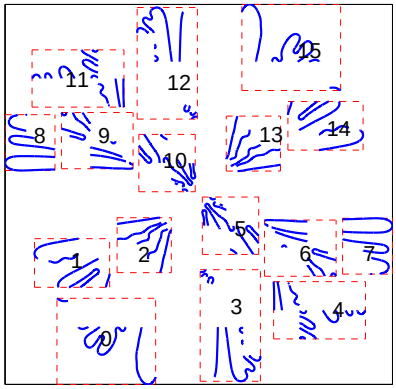
(a) Original precipitate interface



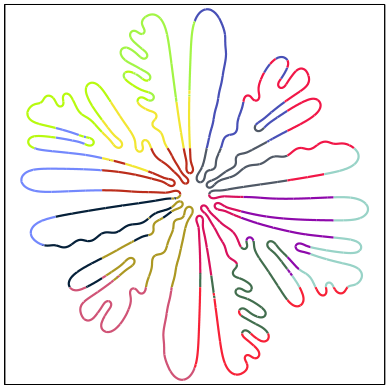
(b) Division by the natural order



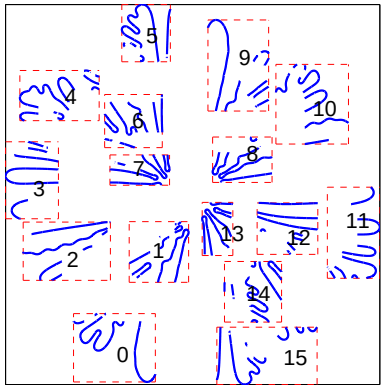
(c) Division by the Morton order



(d) Separation of the left



(e) Division by the Hilbert curve order



(f) Separation of the left

Figure 5: Division of the computational points on the precipitate interface to sixteen processors. (a) shows the original interface; (b) shows the division using the natural order; (c) and (d) show the division using the Morton order with colors and separation respectively; (e) and (f) show the division using the Hilbert curve order with colors and separation respectively.

In the first method, each processor is assigned the same number of points sequential along the interface. The result of this natural order division is shown in Fig. 5(b), where each disjoint piece is assigned to one processor. This method of natural order does not yield a good data locality, because the piece of interface allocated to a processor generally occupy a region that is elongated with a large aspect ratio. In the second method, we order the points using the Z-order curve (Morton order). Each processor is assigned the same number of points, consecutive in Morton order [33], as shown in Fig. 5(c) with colors. We separate the parts there in Fig. 5(d) for a better view. The third method is similar to the second method, but uses a Hilbert curve ordering [27], as shown in Figs. 5(e) and 5(f). To better see the divisions, we bound each region with a rectangle. Unlike division with the natural order where each processor gets a connected set of points, division using the Morton order or the Hilbert curve order could form multiple curves. Between the Morton order and the Hilbert curve order, it is well known that the latter preserves data locality better [27]. As can be seen from the Morton order division in Fig. 5(c), the regions labeled 0 and 3 have points poorly clustered, resulting in slower convergence of Taylor approximations in the treecode. For load balancing, each processor generally will *not* get the same number of the ordered computational points. We will explain this in Section 4.

In the original form, the Morton or Hilbert curve ordering algorithm applies to points with coordinates of integers. We extend this ordering scheme to our computational points on a curve. First, we bound all the computational points with the smallest square, and linearly translate this square region to the first quadrant such that two sides of the square align along the axes. Second, we scale the square together with the computational points inside to a larger square region while fixing the lower left corner at the origin, such that any two points are distinguishable after rounding their coordinates to integers. Then, we round the new coordinates of the computational points to integers. Now, all the points are associated with grid points of integral coordinates, so they can be ordered in the Morton order or the Hilbert curve order using the original scheme. Note that we only use the new integral coordinates of the computational points for the purpose of ordering, and the original coordinates stay unchanged.

4 Results

4.1 Time complexity of the treecode

All computations and tests in this work are performed on a cluster in which each computing node (to distinguish from nodes in the tree structure, we call this computing node) has two physical processor units, and each processor unit is an Intel Xeon E5530 (quad-core) 2.40 GHz QPI with 8MB L2 Cache shared among the four cores. Each computing node has 24GB RAM shared among the eight cores. The computing nodes communicate through InfiniBand connections. In this subsection we document the CPU time of the treecode described in Section 3. To simplify, we use the treecode for the elasticity equations (2.7) and (2.8). In computations of microstructure evolution in elastic media, the

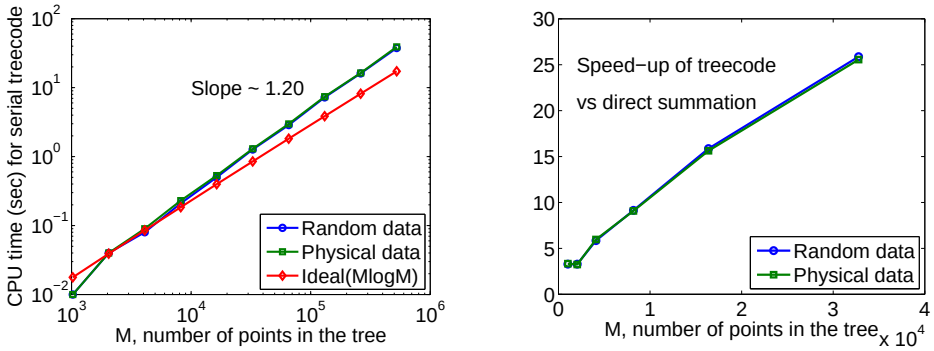
elasticity problem is more time-consuming and involved compared with the diffusion problem, so the elasticity treecode dominates the overall performance of the algorithm.

The treecode is used to compute matrix-vector multiplications in the GMRES solver. We note that we implement an alternating point quadrature when discretizing the integrals in Eqs. (2.7), (2.8), and (2.12), hence two trees are constructed in each GMRES iteration, a tree for even sources (in which the odd points are the targets) and a tree for odd sources (in which even points are targets). For example, if $N=256$ is the number of computational points on the interface, there are $M=128$ points in each tree. The treecode is tested with both physical data and randomly generated data. In the test, we used N ranging from 2^{11} to 2^{20}, with N doubled successively. Equivalently, M ranges from 2^{10} to 2^{19}. In tests using the physical data, for M up to 2^{15}, test data is taken from an evolving interfaces shown in Fig. 7. For $M>2^{15}$, test data is obtained by taking the final interface morphology shown in Fig. 7 and doubling the number of points successively through interpolations. The reason to execute the treecode for larger numbers of points is because, when the physical simulation ends as shown in Fig. 7(f) we only have $M=2^{15}$, which is still relatively small for investigating the asymptotic time complexity of the treecode.

For each M, besides using these physical data, we also tested the treecode with randomly generated data. The random points are uniformly distributed in $(-1,1)\times(-1,1)$, and carry (scalar) weights uniformly distributed in $(-1,1)$. For vector weights like dipoles, each component of the vector is generated with uniform distribution in $(-1,1)$. The random weights are scaled by M before being used for tests.

As explained in Subsection 3.3, there are three user-defined parameters when running the treecode. The first one is the minimum allowable number of points in a cluster N_0. The second one is the maximum allowable order of Taylor approximation $Q_{\max}$. The third one is the error tolerance ϵ. In our tests, we set $N_0=100$, $Q_{\max}=16$, and $\epsilon=10^{-8}$. In GMRES, we use the same error tolerance ϵ for the diffusion and the elasticity problems. As for accuracy, we obtained the running time by averaging over 100 GMRES iterations. The CPU times are plotted in Fig. 6(a) on a log-log scale. We note that for the same number of points, the treecode takes almost the same amount of time for the physical data as that for the random data. This shows that the adaptivity takes care of data non-uniformity in the physical data nicely.

It is difficult to analyze the time complexity of the treecode presented here. The execution time devoted to tree construction is negligible ($<1\%$). If we assume that the moments in Eq. (3.4) are available, the theoretical time complexity is $\mathcal{O}(N\log N)$. However, our tests have shown that moment computation takes a considerable percentage of the CPU time. Different than FMM, which uses an interaction list, the treecode employs a divide-and-conquer strategy, which will also take CPU time. One benefit of the divide-and-conquer strategy is that the treecode can control the absolute error. In Fig. 6(a), we fit the CPU time data of our tests with a straight line using the least square method, and found the slope to be 1.20. In other words, the treecode developed here shows an asymptotic time complexity of $\mathcal{O}(N^{1.2})$. We also performed tests with larger error tolerance, and found the slope to be 1.15 if the error tolerance is relaxed from $\epsilon=10^{-8}$ to $\epsilon=10^{-4}$,



(a) Asymptotic CPU time (b) Speed-up of serial treecode

Figure 6: In (a) is plotted the average CPU time (in seconds) of one GMRES iteration (one matrix-vector multiplication) for the elasticity equation on a single processor as a function of the number of points M in the tree. For reference, the ideal CPU time is also plotted. In (b) is plotted the speedup of the treecode versus the direct summation as a function of the number of points M in the tree. For both plots, the curves with squares and circles are CPU times for the physical and random data respectively, and they are indistinguishable, which shows that the adaptivity of the treecode works well with data non-uniformity.

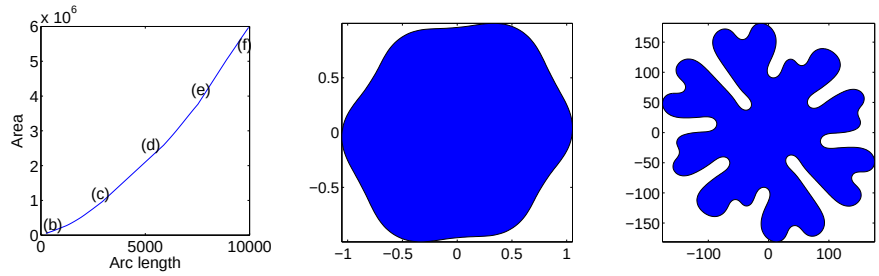
which is not acceptable in our work as we expect spectral accuracy. In terms of practical purposes, an $\mathcal{O}(N^{1.2})$ algorithm is still acceptable. When N increases from 10^3 to 10^6, an $\mathcal{O}(N^{1.2})$ algorithm needs 4,000 times more CPU time, while an $\mathcal{O}(N\log N)$ algorithm needs 2,000 times more CPU time.

In Fig. 6(b), we show the speed-up of treecode against direct summation. The error tolerance of the treecode is 10^{-8}. We note that when $M=32{,}768$, the treecode is about 25 times faster than direct summation.

4.2 Evolution of a single precipitate

The boundary integral method described in Section 2.2 is used to simulate the evolution of a single precipitate in a matrix. To speed up the time evolution, we use a time-rescaling scheme [4, 20]. The initial interface is a perturbed circle with radius $r=1+0.05(\cos 6\theta+\sin 2\theta)$, $\theta\in(0,2\pi)$. The flux is $J=10$, the elasticity coefficient is $Z=20$, and the elastic constants for the matrix and the precipitate are $\mu^P=0.5$, $\nu^P=0.2$, $\mu^M=1.0$, $\nu^M=0.2$. The computation starts with $N=256$ points on the interface. The initial time step is $\Delta t=5\cdot 10^{-4}$. We use the small scale decomposition method in [13] to remove the stiffness of the evolution equations.

To ensure that the interface is well resolved, we double the number of points when the iteration number in GMRES exceeds a threshold. In addition, area and centroid of the precipitate are also monitored as an additional criterion to double the number of computational points. The GMRES tolerance is set at 10^{-8} for both the diffusion and elasticity equations. Two types of filters are used in the computation. The smoothing filter damps out high modes and control aliasing errors [13]. The cut-off filter is used to suppress roundoff errors [17]. The filter level is set to be 10^{-8} for both the smoothing



(a) Area vs. arclength (b) t=0.00 (c) t=0.50

(d) t=0.58 (e) t=0.69 (f) t=0.72

Figure 7: Growth of area relative to the growth of arclength is plotted in (a), with labels corresponding to times shown in (b)-(f). A time sequence of the interface between the precipitate and the matrix is shown in (b)-(f).

filter and the cut-off filter. Every time the number of computational points N is doubled, the time step Δt is halved to satisfy the stability constraints. In the time domain, we use a second order accurate Adams-Bashforth method.

A time sequence of the correspondent interfaces is shown in Fig. 7(b)-(f). In Fig. 7(a), we also plot the area of the precipitate against the arclength of the interface. It shows that the area grows non-linearly in relation to the arclength. To ensure accuracy of the computation, we monitor the scaled precipitate area during the computation. When the computation ends, the area has six accurate digits. Another indication of accuracy is to check the two-fold axial symmetry of the evolution for the initial precipitate shape used. At early times, the morphology appears to be purely diffusional with no preference for directions. This is a stage in which area grows slowly relative to arclength growth, as shown by the first segment in Fig. 7(a). At late times, the effect of elasticity becomes more prominent and results in two changes. Namely, the precipitate starts to align in the x direction and the y direction due to the applied elastic boundary condition, and the growth of the precipitate area is faster than that at early times.

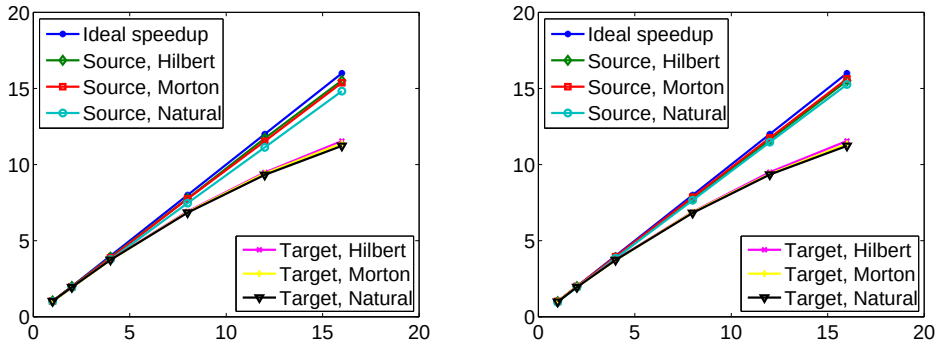
4.3 Speedup of the parallel treecode over serial version

For parallel implementation of the treecode, we compare the source dividing strategy and the target dividing strategy described in Subsection 3.4. We only test the treecodes for the elasticity problem since they are more time-consuming compared with the diffusion

problem. We use the physical data shown in Fig. 5(a). We also test the treecodes with randomly generated data. Note that since an alternating point quadrature is used, the target points and the source points are distinct sets. In the serial code, there are two trees, the target and source point sets are one of each other in the two trees. For the purpose of studying speedup of the parallel treecode, it suffices to parallelize just one of the two trees. When the simulation of a single precipitate morphology gets to the stage as shown in Fig. 5(a), it takes about 150 iterations for the GMRES solver to converge if the error tolerance is 10^{-8}. As before, we measure CPU times by averaging 100 GMRES iterations. We measure the scaling efficiency of the treecode by strong scaling law, i.e., fixing the problem size while varying the number of processors.

Between the GMRES iterations in a time step, the locations of the computational points do not change, only the weights (single layer strength, double layer strength, etc.) associated with these points change. The significance of this is that once the points are ordered, the ordering can be used in all GMRES iterations in the same time step. At the beginning of each time step, the computational points are ordered by the head processor using one of the three methods as described in Subsection 3.4, and then these points are assigned to processors evenly. For each of the following GMRES iterations, the head processor first weighs the points by the average CPU time that their host processor spends on them, and then reassign them to processors for the purpose of load balancing.

We next compare the target dividing strategy and the source dividing strategy. For the target dividing strategy, each processor forms its own tree from all the source points and computes moments themselves. This makes the two strategies comparable in terms of programming complexity. In Fig. 8, we plot speedup of the parallel treecode against the number of processors. The parallel treecode with the source dividing strategy shows



(a) physical data (b) random data

Figure 8: Speedup of the parallel treecode as a function of the number of processors. The horizontal axis is the number of processors M, and the vertical axis is the speedup defined as serial code / parallel code in terms of CPU time. The physical data is used in (a), and the random data is used in (b). In each subfigure, we plot measurements from parallel treecode using both the target dividing strategy and the source dividing strategy, and for each strategy, we test all the three methods of division as explained in Subsection 3.4.

an almost linear speedup for up to sixteen processors, while the parallel treecode with the source dividing strategy loses linear speedup when the number of processors goes from six to eight.

We also tested the parallel treecode with source dividing strategy on 32 processors, but the speedup of the parallel treecode falls to about 25, amounting to a parallelization efficiency of 78%. We believe the reason is as follows. When the 2D treecode runs parallel on sixteen processors, each processor essentially contains a subtree enclosing computational points in a region whose radius is approximately $1/\sqrt{16}=1/4$ of the radius of the entire computational domain. With more processors, source points assigned to a processor will occupy an even smaller space. The result is that during the divide-and-conquer process, the error tolerance may allow a point-cluster interaction between a target point and a source cluster consisting of points assigned to different processors, and hence optimal performance can not be achieved. For many problems of interest, an almost linear speedup with sixteen processors is still acceptable. An immediate future project would be to compare our source dividing strategy to the best existing parallel treecode.

Acknowledgments

This work was supported by the NSF through Grant DMS-0923111, DMS-0914923, and an Illinois Institute of Technology post-doctoral traveling allowance. The authors would also like to thank Prof. Jingfang Huang from University of North Carolina, Dr. H.-J. Jou from QuesTek Innovations LLC., and Prof. Qing Nie from University of California-Irvine for helpful discussions.

References

[1] N. Akaiwa and D.I. Meiron. Numerical-simulation of 2-dimensional late-stage coarsening for nucleation and growth. Physical Review E, 51:5408, 1995.
[2] J. Barnes and P. Hut. A hierarchical $\mathcal{O}(N\log N)$ force-calculation algorithm. Nature, 324(6096):446–449, 1986.
[3] A. Barua, S. Li, H. Feng, X. Li, and J. Lowengrub. An efficient rescaling algorithm for simulating the evolution of multiple elastic precipitates. Commun. Comput. Phys., 14(4):940–959.
[4] A. Barua, S. Li, X. Li, and J. Lowengrub. Self-similar evolution of a precipitate in inhomogeneous elastic media. J. Cryst. Growth, 351(1):62–71, 2012.
[5] T. Belytschko, R. Gracie, and G. Ventura. A review of extended/generalized finite element methods for material modeling. Mod. Simu. in Mat. Sci. Engi., 17(4), 2009.
[6] J. Boisse, N. Lecoq, R. Patte, and H. Zapolsky. Phase-field simulation of coarsening of γ precipitates in an ordered γ' matrix. Acta Materialia, 55(18):6151–6158, 2007.
[7] G.F. Carrier, M. Krook, and C.E. Pearson. Function of a Complex Variable. McGraw-Hill, 1966.
[8] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. J. Comput. Phys., 155(2):468–498, 1999.

[9] C.I. Draghicescu and M. Draghicescu. A fast algorithm for vortex blob interactions. J. Comput. Phys., 116(1):69–78, 1995.
[10] A. Greenbaum, L. Greengard, and G. B. McFadden. Laplace equation and Dirichlet-Neumann map in multiply connected domains. J. Comput. Phys., 105:267, 1993.
[11] L. Greengard and W. Gropp. A parallel version of the fast algorithm method. Comp. Math. Appl., 20(7):63–71, 1990.
[12] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. J. Comput. Phys., 73(2):325–348, 1987.
[13] T. Hou, J.S. Lowengrub, and M. Shelley. Removing the stiffness from interfacial flows with surface tension. J. Comput. Phys., 114:312, 1994.
[14] T. Hou, J.S. Lowengrub, and M. Shelley. Boundary integral methods for multicomponent fluids and multiphase materials. J. Comput. Phys., 169:302, 2001.
[15] H.-J. Jou, P.H. Leo, and J.S. Lowengrub. Microsructural evolution in inhomogeneous elastic media. J. Comput. Phys., 131:109, 1997.
[16] I. Kabadshow, H. Dachsel, and J. Hammond. Poster: Passing the three trillion particle limit with an error-controlled fast multipole method. In SC Companion, pages 73–74, 2011.
[17] R. Krasny. A study of singularity formation in a vortex sheet by the point-vortex approximation. J. Fluid Mech., 167:65–93, 1986.
[18] Y. Kwon, K. Thornton, and P. W. Voorhees. Morphology and topology in coarsening of domains via non-conserved and conserved dynamics. Philosophical Magazine, 90(1-4):317–335, 2010.
[19] B. Li, J. Lowengrub, A. Raetz, and A. Voigt. Geometric evolution laws for thin crystalline films: modeling and numerics. Commun. Comput. Phys., 6(3):433–482, 2009.
[20] S. Li, J. Lowengrub, and P. H. Leo. A rescaling scheme with application to the long-time simulation of viscous fingering in a hele-shaw cell. J. Comput. Phys., 225(1):554–567, 2007.
[21] K. Lindsay and R. Krasny. A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow. J. Comput. Phys., 172(2):879–907, 2001.
[22] D. Liu, Z.H. Duan, R. Krasny, and J. Zhu. Parallel implementation of the treecode Ewald method. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, Sante Fe, New Mexico, 2004.
[23] S.G. Mikhlin. Integral equation and their applications to certian problems of mechanics. Pergammon, New York, 1957.
[24] R. S. Qin and H. K. Bhadeshia. Phase field method. Mat Sci and Tech, 26(7):803–811, 2010.
[25] A. Rahimian, I. Lashuk, S. Veerapaneni, C. Aparna, D. Malhotra, I. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In ACM/IEEE SCxy conference series, pages 1–11, 2010.
[26] Y. Saad and M. Schultz. A generalized minimum residual method for solving non symmetric linear systems. SIAM J. Sci. Stat. Comput., 7:856, 1986.
[27] H. Samet. The Design and Analysis of Spatial Data Structures. Addison-Wesley, New York, 1989.
[28] A. Sidi and M. Israel. Quadrature methods for periodic singular and weakly singular Fredholm integral equations. J. Sci. Comput., 3:201, 1988.
[29] J.P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in adaptive hierarchical N-body methods – Barnes-Hut, fast multipole, and radiosity. J. Parallel Distrib. Comput., 27(2):118–141, 1995.
[30] I. Steinbach. Phase-field models in materials science. Mod. Simu. in Mat. Sci. Engi., 17(7),

2009.
[31] K. Thornton, N. Akaiwa, and P.W. Voorhees. Large-scale simulations of Ostwald ripening in elastically stressed solids: I. Development of microstructure. Physical Review E, 52:1353, 2004.
[32] K. Thornton, J. Argen, and P. W. Voorhees. Modeling the evolution of phase boundaries in solids at the meso- and nano-scales. Acta Materilia, 51(19):5675–5710, 2003.
[33] M.S. Warren and J.K. Salmon. A portable parallel particle program. Comput. Phys. Comm., 87:266–290, 1995.
[34] M. Winkel, R. Speck, H. Hubner, L. Arnold, R. Krause, and P. Gibbon. A massively parallel, multi-disciplinary Barnes-Hut tree code for exterme-scale N-body simulations. Comput. Phys. Comm., 183:880–889, 2012.
[35] R. Yokota, L. Barba, and M. Knepley. PetRBF – A parallel $\mathcal{O}(N)$ algorithm for radial basis function interpolation. Comput. Meth. Appl. Mech. Eng., 199(25-28):1793–1804, 2010.
[36] B. Zhang, J. Huang, N. Pitsianis, and X. Sun. Dynamic prioritization for parallel traversal of irregularly structured spatio-temporal graphs. In Proceedings of the 3rd USENIX Workshop on Hot Topics in Parallelism, Berkeley, CA, 2011.