Comparison of Treecodes for Computing Electrostatic Potentials in Charged Particle Systems with Disjoint Targets and Sources

Henry A. Boateng*† and Robert Krasny*

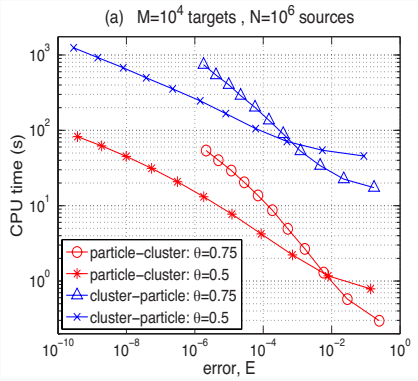
June 8, 2013

Abstract

In molecular simulations it is sometimes necessary to compute the electrostatic potential at M target sites due to a disjoint set of N charged source particles. Direct summation requires $O(MN)$ operations, which is prohibitively expensive when M and N are large. Here we consider two alternative tree-based methods which reduce the cost. The standard particle-cluster treecode partitions the N sources into an octree and applies a far-field approximation, while a recently developed cluster-particle treecode instead partitions the M targets into an octree and applies a near-field approximation. We compare the two treecodes with direct summation and document their accuracy, CPU run time, and memory usage. We find that the particle-cluster treecode is faster when $N > M$, i.e. when the sources outnumber the targets, and conversely, the cluster-particle treecode is faster when $M > N$, i.e. when the targets outnumber the sources. Hence the two treecodes provide useful tools for computing electrostatic potentials in charged particle systems with disjoint targets and sources.

Keywords: charged particle system, electrostatic potential, fast summation, treecode, molecular simulation

*Department of Mathematics, University of Michigan, Ann Arbor, Michigan 48109
†boateng@umich.edu



Two alternative fast summation treecodes are described for computing electrostatic potentials in charged particle systems with disjoint targets and sources. We compare the treecodes with direct summation and document their accuracy, CPU run time, and memory usage. The particle-cluster treecode is faster when the sources outnumber the targets, and conversely, the cluster-particle treecode is faster when the targets outnumber the sources.

INTRODUCTION

Charged particle interactions are a key element in molecular simulations[1,2]. In a system with N charged particles, evaluating the electrostatic potential at the particle locations by direct summation requires $O(N^2)$ operations, which is prohibitively expensive when N is large. As a result, many approaches have been developed to reduce the cost and permit the study of larger systems. For example, particle-mesh methods project the particles onto a regular grid and apply the Fast Fourier Transform to evaluate the potential. More efficient related methods include particle-particle/particle-mesh[3] and particle-mesh Ewald[4–6]. Another approach using interpolation onto multiple nested grids is the multilevel summation method[7–10].

An alternative class of tree-based methods use particle clustering and analytic approximations to reduce the cost. This class includes the treecode[11] and the fast multipole method (FMM)[12–14]. In a treecode, the particles are divided into an octree of clusters and the particle-particle interactions are replaced by particle-cluster interactions. Well-separated interactions are evaluated using a far-field multipole approximation[15] and the remaining interactions are evaluated directly. We refer to this approach as a particle-cluster treecode. In the FMM, the multipole approximations at different levels of the tree are combined and evaluated by a local approximation at the leaves.

Particle-mesh and tree-based methods reduce the operation count to $O(N \log N)$ and hence they are heavily used in molecular simulations. Nonetheless, there is ongoing interest in extending the capability of these methods and further optimizing their performance.

Here we consider the problem of evaluating the electrostatic potential at M target sites due to a disjoint set of N charged source particles, where $M \neq N$. One potential application is in particle-mesh simulations, for example as depicted in Figure 1, with random charged source particles representing bulk liquid and target sites on a grid. A special case arises in mesh-based finite-difference solutions of the Poisson-Boltzmann equation for solvated biomolecules, where the sources are the atomic charges representing the biomolecule and the targets are the boundary points of the finite-difference grid. In this case the high cost of evaluating the Dirichlet boundary values of the potential is a computational bottleneck[16–18], and the approach described here may be able to reduce the cost. Another

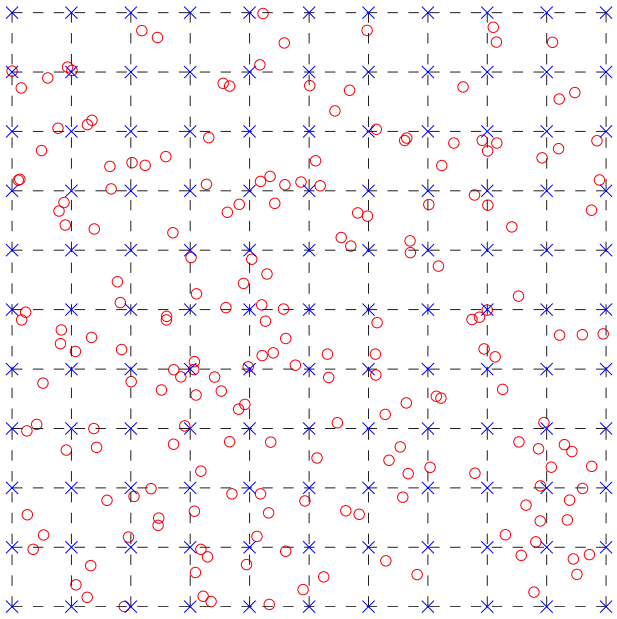


Figure 1: Schematic picture of a particle-mesh simulation, random charged source particles (○) representing bulk liquid, target sites (x) on a grid.

potential application arises when the targets and sources represent two distinct molecules, as in protein docking simulations.

To set notation, consider a set of target sites $\{\mathbf{x}_i, i = 1 : M\}$ and a set of source particles with associated partial charges $\{\mathbf{y}_j, q_j, j = 1 : N\}$, where $\mathbf{x}_i, \mathbf{y}_j \in \mathbb{R}^3$. We consider potentials of the form

$$V(\mathbf{x}_i) = \sum_{j=1}^{N} q_j \phi(\mathbf{x}_i, \mathbf{y}_j), \quad i = 1 : M, \tag{1}$$

where $\phi(\mathbf{x}, \mathbf{y})$ is a given kernel, and the goal is to compute these quantities accurately and efficiently. We present results for the Coulomb potential,

$$\phi(\mathbf{x}, \mathbf{y}) = \frac{1}{4\pi\epsilon_0 |\mathbf{x} - \mathbf{y}|}, \tag{2}$$

but several other kernels can be treated similarly including the real space Ewald potential[19], power law potential[20], the screened Coulomb potential[21] and the generalized Born potential[22].

Direct summation of Equation (1) using loops over i and j has operation count $O(MN)$ and the need arises for more efficient methods when M or N are large. Unlike the case of coincident targets and sources, the case of disjoint targets and sources has received relatively

little attention. In one example from computational fluid dynamics, the FMM was used to compute the stream function at a set of targets due to disjoint sources in a two-dimensional flow field[23].

The present work considers two alternative tree-based methods for computing the potentials in Equation (1), the standard particle-cluster treecode[11,20,24] and a recently developed cluster-particle treecode[25]. The particle-cluster treecode partitions the N sources into an octree and applies a far-field approximation, while the cluster-particle treecode instead partitions the M targets into an octree and applies a near-field approximation. The cluster-particle treecode was introduced in the context of radial basis function approximations[25], while the present work emphasizes the application to electrostatics, especially for $M \neq N$.

We compare the two treecodes with direct summation and document their accuracy, CPU run time, and memory usage. Depending on the required error, the treecodes run at a fraction of the time taken by direct summation, and their memory usage is a relatively modest factor more than the direct sum memory usage. We find that the particle-cluster treecode is faster when $N > M$, i.e. when the sources outnumber the targets, and conversely, the cluster-particle treecode is faster when $M > N$, i.e. when the targets outnumber the sources. Hence the two treecodes provide useful tools for computing electrostatic potentials in charged particle systems with disjoint targets and sources.

The remainder of the article is organized as follows. The methodology is explained in the next section including the procedure for constructing the tree, the particle-cluster and cluster-particle algorithms, the recurrence relation used for the near-field and far-field approximations, and some implementation details. This is followed by numerical results for two test cases in which we present the error, CPU run time, and memory usage, showing how the treecodes' performance depends on the system size M, N and the user-specified treecode parameters. This is followed by concluding remarks.

METHODOLOGY

Tree construction

Consider a set of particles representing either M targets or N sources. The root cluster is the smallest rectangular box enclosing the particles, with sides parallel to the Cartesian axes. The root is divided uniformly into eight child clusters forming the next level in the tree. The process continues recursively until the number of particles in a cluster is less than a user-specified value (M_0 for a target tree, N_0 for a source tree). This yields an octree of particle clusters[11]. A cluster at the lowest level of the tree is called a leaf. The number of levels depends logarithmically on the number of particles. In the following sections we describe the two versions of the treecode.

Particle-cluster treecode

Particle-cluster treecodes have been applied mainly to systems in which the targets and sources coincide[11,20–22,24], but the approach extends readily to systems with disjoint targets and sources. In this case the tree construction procedure is applied to the N source particles $\{\mathbf{y}_j\}$, yielding an octree of source clusters. The potential is written as a sum of particle-cluster interactions,

$$V(\mathbf{x}_i) = \sum_{j=1}^{N} q_j \phi(\mathbf{x}_i - \mathbf{y}_j) \tag{3a}$$

$$= \sum_{C} \sum_{\mathbf{y}_j \in C} q_j \phi(\mathbf{x}_i, \mathbf{y}_j), \tag{3b}$$

where the source clusters C in Equation (3b) depend on the target site $\mathbf{x}_i$ and are determined in a manner described below.

Figure 2 depicts a particle-cluster interaction between a target site $\mathbf{x}_i$ and the source particles $\mathbf{y}_j$ in source cluster C, showing the cluster center $\mathbf{y}_c$, cluster radius r, and particle-cluster distance $R = |\mathbf{x}_i - \mathbf{y}_c|$. The algorithm employs a far-field approximation which is valid when the target site $\mathbf{x}_i$ is far from the source cluster C.

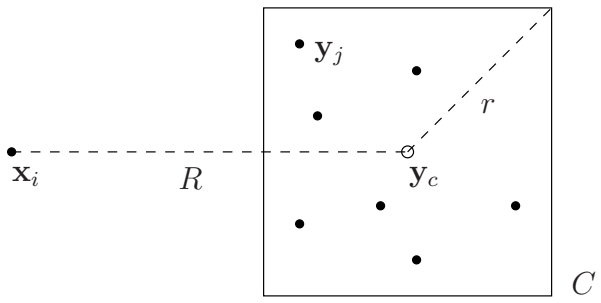


Figure 2: A particle-cluster interaction is depicted between a target site $\mathbf{x}_i$ and the source particles $\mathbf{y}_j$ in source cluster C. The cluster has center $\mathbf{y}_c$ and radius r, and the particle-cluster distance is R.

A particle-cluster interaction can be computed by direct summation or by Taylor expanding $\phi(\mathbf{x}_i, \mathbf{y}_j)$ about $\mathbf{y} = \mathbf{y}_c$,

$$\sum_{\mathbf{y}_j \in C} q_j \phi(\mathbf{x}_i, \mathbf{y}_j) \approx \sum_{\mathbf{y}_j \in C} q_j \sum_{||\mathbf{k}||=0}^{p} \frac{1}{\mathbf{k}!} \partial_{\mathbf{y}}^{\mathbf{k}} \phi(\mathbf{x}_i, \mathbf{y}_c)(\mathbf{y}_j - \mathbf{y}_c)^{\mathbf{k}} \tag{4a}$$

$$= \sum_{||\mathbf{k}||=0}^{p} a_{\mathbf{k}}(\mathbf{x}_i, \mathbf{y}_c) M_{\mathbf{k}}(C). \tag{4b}$$

In Equation (4a) we used Cartesian multi-index notation, $||\mathbf{k}|| = k_1 + k_2 + k_3$, $\mathbf{k}! = k_1! k_2! k_3!$, $\partial_{\mathbf{y}}^{\mathbf{k}} = \partial y_1^{k_1} \partial y_2^{k_2} \partial y_3^{k_3}$, $(\mathbf{y}_j - \mathbf{y}_c)^{\mathbf{k}} = (y_{j_1} - y_{c_1})^{k_1} (y_{j_2} - y_{c_2})^{k_2} (y_{j_3} - y_{c_3})^{k_3}$, and in Equation (4b) we defined the Taylor coefficients of the kernel,

$$a_{\mathbf{k}}(\mathbf{x}_i, \mathbf{y}_c) = \frac{1}{\mathbf{k}!} \partial_{\mathbf{y}}^{\mathbf{k}} \phi(\mathbf{x}_i, \mathbf{y}_c), \tag{5}$$

and the cluster moments

$$M_{\mathbf{k}}(C) = \sum_{\mathbf{y}_j \in C} q_j (\mathbf{y}_j - \mathbf{y}_c)^{\mathbf{k}}. \tag{6}$$

Equation (4b) defines a pth order Taylor approximation for the particle-cluster interaction.

Algorithm 1 describes the particle-cluster treecode[11,21]. The user-specified parameters are p (order of Taylor approximation), N_0 (maximum number of particles in a leaf), and θ (defined below). The code loops over the target sites $\mathbf{x}_i$ to compute the potentials $V(\mathbf{x}_i)$. A recursive subroutine **compute-pc** is called to compute the interaction between a target site and a source cluster. A target site and source cluster are called well-separated if $r/R \leq \theta$ and this defines the multipole acceptance criterion (MAC)[11,26]. If the MAC is satisfied, then

the interaction is computed by the Taylor approximation in Equation (4b). If the MAC is not satisfied, then the code descends to the children of the cluster, until a leaf is reached at which point direct summation is used. The cluster moments are computed on the fly and stored for re-use with different target sites. The Taylor coefficients are computed using a recurrence relation given below.

Algorithm 1. Particle-cluster treecode

```
1  program pc-treecode
2     input : targets x_i, sources and charges y_j, q_j, treecode parameters p, θ
3     output : potentials V(x_i)
4     construct tree of source clusters
5     for i = 1 : M; compute-pc(x_i,root); end
6  end program
7  subroutine compute-pc (x, C)
8     if MAC is satisfied
9        compute and store moments of C (if not already available)
10       compute particle-cluster interaction by Taylor approximation (4b)
11    else if C is a leaf
12       compute particle-cluster interaction by direct summation
13    else for each child of C
14       compute-pc(x,child)
15 end subroutine
```

In the particle-cluster treecode, the tree has $O(\log N)$ levels. The operation count for the cluster moments is $O(N \log N)$, since each source particle contributes to a cluster on every level of the tree. The operation count for computing the potentials is $O(M \log N)$, since the code loops over the targets and descends to the leaves of the tree in each step of the loop. Note that the operation count for the Taylor approximation (4b) is independent of M and N, assuming the moments are known. In addition we are not accounting for N_0, the maximum number of particles in a leaf. Hence according to these considerations, the particle-cluster treecode has operation count $O((M + N) \log N)$.

Cluster-particle treecode

In this case the tree construction procedure is applied to the M target sites $\{\mathbf{x}_i\}$, yielding an octree of target clusters. Figure 3 depicts a cluster-particle interaction between the target sites $\mathbf{x}_i$ in a target cluster C and a source particle $\mathbf{y}_j$. The cluster has center $\mathbf{x}_c$ and radius r, and the cluster-particle distance is $R = |\mathbf{x}_c - \mathbf{y}_j|$. The target cluster and source particle are called well-separated if $r/R \leq \theta$. The algorithm employs a near-field approximation which is valid for all target sites $\mathbf{x}_i \in C$ and which accounts for interactions with well-separated source particles $\mathbf{y}_j$.

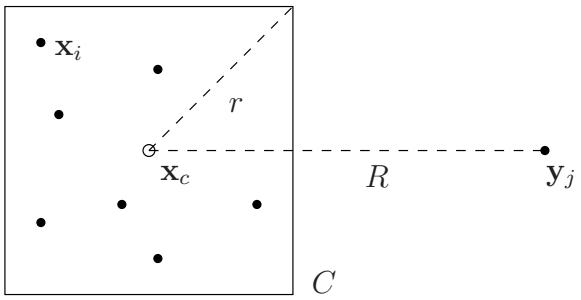


Figure 3: A cluster-particle interaction is depicted between the target sites $\mathbf{x}_i$ in a target cluster C and a source particle $\mathbf{y}_j$. The cluster has center $\mathbf{x}_c$ and radius r, and the cluster-particle distance is R.

Suppose the tree has L levels, where level 1 is the root and level L contains the leaves. A given target site $\mathbf{x}_i$ belongs to a nested sequence of clusters, $\mathbf{x}_i \in C_L \subseteq \cdots \subseteq C_1$, where the subscript denotes the level of the cluster in the tree. The center of cluster C_l is denoted by $\mathbf{x}_c^l$ for $1 \leq l \leq L$. Now let I_l be the set of source particles $\mathbf{y}_j$ that are well-separated from C_l but are not well-separated from $C_{l-1}, \ldots, C_1$, and let D be the set of source particles $\mathbf{y}_j$ that are not well-separated from $C_L, \ldots, C_1$. This effectively divides the source particles into a set of interaction lists. Then the potential at $\mathbf{x}_i$ can be expressed as

$$V(\mathbf{x}_i) = \sum_{j=1}^{N} q_j \phi(\mathbf{x}_i, \mathbf{y}_j) \tag{7a}$$

$$= \sum_{\mathbf{y}_j \in D} q_j \phi(\mathbf{x}_i, \mathbf{y}_j) + \sum_{l=1}^{L} \sum_{\mathbf{y}_j \in I_l} q_j \phi(\mathbf{x}_i, \mathbf{y}_j). \tag{7b}$$

The first term in Equation (7b) is evaluated by direct summation and the remaining terms are evaluated by Taylor expanding $\phi(\mathbf{x}_i, \mathbf{y}_j)$ about $\mathbf{x} = \mathbf{x}_c^l$ to obtain

$$\sum_{\mathbf{y}_j \in I_l} q_j \phi(\mathbf{x}_i, \mathbf{y}_j) \approx \sum_{\mathbf{y}_j \in I_l} q_j \sum_{||\mathbf{k}||=0}^{p} \frac{1}{k!} \partial_{\mathbf{x}}^{\mathbf{k}} \phi(\mathbf{x}_c^l, \mathbf{y}_j)(\mathbf{x}_i - \mathbf{x}_c^l)^{\mathbf{k}} \tag{8a}$$

$$= \sum_{||\mathbf{k}||=0}^{p} m_{\mathbf{k}}(\mathbf{x}_c^l)(\mathbf{x}_i - \mathbf{x}_c^l)^{\mathbf{k}}. \tag{8b}$$

Equation (8b) is a power series of degree p in the variable $\mathbf{x}_i - \mathbf{x}_c^l$ with coefficients

$$m_{\mathbf{k}}(\mathbf{x}_c^l) = \sum_{\mathbf{y}_j \in I_l} q_j (-1)^{||\mathbf{k}||} a_{\mathbf{k}}(\mathbf{x}_c^l, \mathbf{y}_j), \tag{9}$$

where we used the definition of the Taylor coefficients in Equation (5) and the relation $\partial_{\mathbf{x}}^{\mathbf{k}} \phi = (-1)^{||\mathbf{k}||} \partial_{\mathbf{y}}^{\mathbf{k}} \phi$. Equation (8b) defines a pth order Taylor approximation for the cluster-particle interaction between target sites $\mathbf{x}_i \in C_l$ and source particles $\mathbf{y}_j \in I_l$. We view it as a near-field approximation since it is valid when $\mathbf{x}_i$ is close to $\mathbf{x}_c^l$.

The cluster-particle treecode is described in Algorithm 2. First the tree construction procedure is applied to the M target sites to create the target clusters. Then there are two stages. Stage 1 computes the power series coefficients $m_{\mathbf{k}}(\mathbf{x}_c)$ and the direct sum term in Equation (7b). Stage 2 computes the remaining terms in Equation (7b) using the near-field approximation in Equation (8b).

In stage 1 the code loops through the N source particles using subroutine **compute-cp1**, and for each source particle $\mathbf{y}_j$, it descends the tree of target clusters starting from the root. If a target cluster and the source particle are well-separated, then the power series coefficients $m_{\mathbf{k}}(\mathbf{x}_c)$ in Equation (9) are updated. Otherwise the subroutine descends to the children of the cluster and calls itself recursively. When a leaf C_L is reached, if the MAC is not satisfied, then the first term in Equation (7b) is computed by direct summation for all target sites $\mathbf{x}_i \in C_L$.

In stage 2 the code uses subroutine **compute-cp2** to compute the remaining terms in Equation (7b). This is done by descending the tree and evaluating the power series in Equation (8b) for all target sites $\mathbf{x}_i$ in target clusters C that interacted with source particles $\mathbf{y}_j$ by Taylor approximation in stage 1. This completes the evaluation of the potentials $V(\mathbf{x}_i)$.

Algorithm 2. Cluster-particle treecode

```
1  program cp-treecode
2    input : targets x_i, sources and charges y_j, q_j, treecode parameters p, θ
3    output : potentials V(x_i)
4    construct tree of target clusters
5    for j = 1, N; compute-cp1(root,y_j); end
6    compute-cp2(root)
7  end program
8  subroutine compute-cp1(C, y)
9    if MAC is satisfied
10     update power series coefficients m_k(x_c) by Equation (9)
11   else if C is a leaf
12     compute first term in Equation (7b) by direct summation
13   else for each child of C
14       compute-cp1(child,y)
15 end subroutine
16 subroutine compute-cp2(C)
17   if C interacted with a source particle by Taylor approximation in stage 1
18     loop through target sites x_i in C
19     compute second term in Equation (7b) using power series Equation (8b)
20   for each child of C
21     compute-cp2(child)
22 end subroutine
```

In the cluster-particle treecode, the tree has $O(\log M)$ levels and the code descends through the tree in both stages. In stage 1 the code loops through the source particles so the operation count is $O(N \log M)$, and in stage 2 the code accesses the target sites so the operation count is $O(M \log M)$. Note that the operation count for the power series (8b) is independent of M and N, assuming the coefficients are known. In addition we are not accounting for M_0, the maximum number of particles in a leaf. Hence according to these

considerations, the cluster-particle treecode has operation count $O((M+N)\log M)$.

Recurrence relation

The recurrence relation[19,24] for the Taylor coefficients $a_{\mathbf{k}}(\mathbf{x},\mathbf{y})$ of the Coulomb potential is

$$a_{\mathbf{k}} = \frac{1}{|\mathbf{x}-\mathbf{y}|^2}\left[\left(\frac{1}{||\mathbf{k}||}-2\right)\sum_{i=1}^{3}(x_i-y_i)a_{\mathbf{k}-\mathbf{e}_i} + \left(\frac{1}{||\mathbf{k}||}-1\right)\sum_{i=1}^{3}a_{\mathbf{k}-2\mathbf{e}_i}\right], \qquad (10)$$

where $\mathbf{e}_i$ is the ith Cartesian basis vector. The coefficients for $||\mathbf{k}|| = 0, 1$ are computed explicitly and the recurrence relation is used to compute the coefficients for $||\mathbf{k}|| \geq 2$. The procedure assumes that $a_{\mathbf{k}} = 0$ when any index is negative. The particle-cluster treecode uses $(\mathbf{x},\mathbf{y}) = (\mathbf{x}_i,\mathbf{y}_c)$ and the cluster-particle treecode uses $(\mathbf{x},\mathbf{y}) = (\mathbf{x}_c,\mathbf{y}_j)$.

Implementation details

The algorithms were programmed in Fortran90 starting from an open source particle-cluster treecode[27] and the codes are available online[28] under the GNU General Public License. The particle positions and charges are stored in arrays, and a data structure holds information about the clusters, e.g. spatial extent, pointers to the particles contained in the cluster, pointers to the children, and cluster moments or power series coefficients. The computations were performed on an iMAC 2.5GHz quad-core Intel Core i5 processor with 4GB memory running OS X version 10.6.8, and the codes were compiled using ifort with -fast optimization. Memory usage was obtained from the Real Mem column of the OS X Activity Monitor.

Following are some details relevant to the cluster-particle treecode. The power series in Equation (8b) is evaluated efficiently by Horner's rule[29]. The idea can be illustrated simply for a quadratic polynomial in one dimension, $c_0+c_1x+c_2x^2 = c_0+x(c_1+c_2x)$; the expression on the left requires three multiplications and two additions, while the expression on the right requires one less multiplication. This can be generalized to higher degree multi-dimensional power series as in Equation (8b). In stage 1, when the MAC is satisfied for a given target cluster, a flag is set to indicate that a contribution from that cluster will be required in stage 2.

RESULTS

The particle-cluster and cluster-particle treecodes were applied to compute the electrostatic potential in Equation (1) at M target sites due to N disjoint source particles. We considered two test cases. In case 1, the targets and sources are randomly distributed in a unit cube, and in case 2, the targets and sources are chosen to represent a particle-mesh computation. In both cases the source charges q_j are randomly distributed in the interval $(-1, 1)$. The treecode error is defined by

$$E = \left(\frac{\sum_{i=1}^{M} |V(\mathbf{x}_i) - \widehat{V}(\mathbf{x}_i)|^2}{\sum_{i=1}^{M} |V(\mathbf{x}_i)|^2} \right)^{1/2}, \tag{11}$$

where $V(\mathbf{x}_i)$ is the exact potential obtained by direct summation and $\widehat{V}(\mathbf{x}_i)$ is the treecode approximation. The treecode approximation order spanned the range $p = 2k$ for $k = 0, \ldots, 10$, and the maximum number of particles in a leaf was set to $M_0 = N_0 = 500$. The MAC parameter $\theta = 0.75$ is considered first and later on we compare with $\theta = 0.5$. These parameter values are intended as representative rather than optimal values.

Treecode performance in test case 1

In test case 1 the targets and sources are randomly distributed in a unit cube. Figure 4 displays the treecode performance in terms of the error and CPU run time. The number of targets is $M = 10^4, 10^5, 10^6$ from left to right, and the number of sources in each frame is $N = 10^4, 10^5, 10^6$. The MAC parameter is $\theta = 0.75$.

In Figure 4a we see that the treecode error decreases as the approximation order p increases. Order $p = 0$ yields the largest error with $E \approx 10^{-1}$ and order $p = 20$ yields the smallest error with $E \approx 10^{-6}$. For a given order p, both treecodes yield similar errors and the error is almost independent of the system size M, N.

Next consider the CPU run time in Figure 4b. The direct sum CPU time is shown as a dashed line. For a given value of M, the direct sum CPU time increases linearly with N, and conversely, for a given value of N, the direct sum CPU time increases linearly with M;

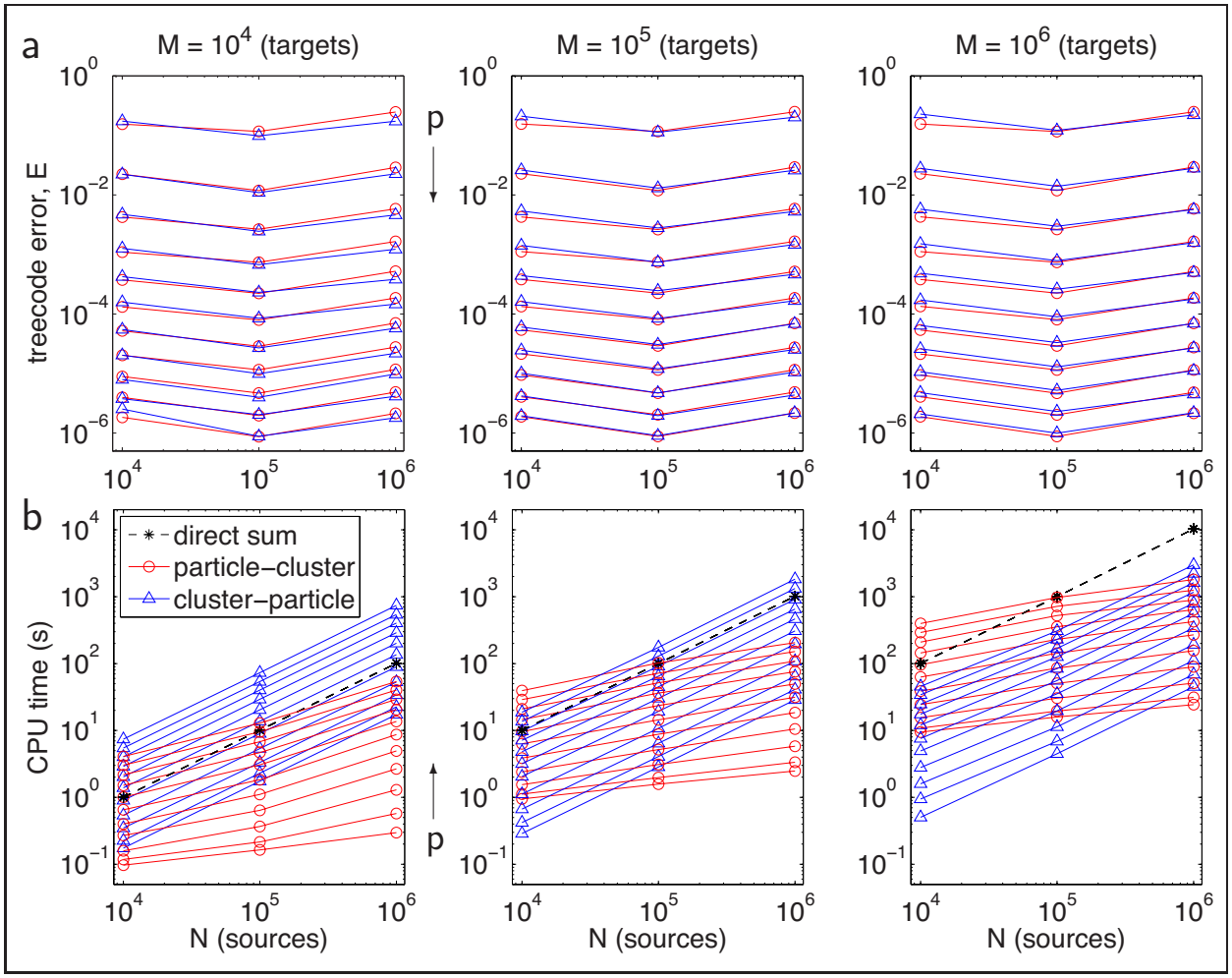


Figure 4: Test case 1, treecode performance, (a) error, (b) CPU time, M targets and N sources, order $p = 2k, k = 0, \ldots, 10$ (arrows indicate increasing p), MAC parameter $\theta = 0.75$, direct sum ($*$, dashed line), particle-cluster ($\bigcirc$, solid line), cluster-particle ($\triangle$, solid line).

to make the latter point more clearly, Table 1a displays the direct sum CPU time. These results confirm as expected that the direct sum CPU time scales like $O(MN)$.

Continuing in Figure 4b, the treecode CPU time is shown as a solid line, with symbols and color distinguishing between particle-cluster and cluster-particle. The treecode CPU time increases as the order p increases, but in many instances the treecodes are faster than direct summation. This is also evident in Table 1b,c which displays the treecode CPU time with order $p = 4$, yielding error $E \approx 5 \cdot 10^{-3}$. For example when $M = N = 10^6$, direct summation takes 10345.48 s $\approx$ 2.9 hr, while the particle-cluster treecode takes 51 s, a factor of 200 times faster.

Next we note that the two treecodes behave differently when $M \neq N$. For example in Table 1 when $M = 10^4, N = 10^6$, particle-cluster takes 1.29 s while cluster-particle takes 34 s, so particle-cluster is faster when the sources outnumber the targets. Conversely, when $M = 10^6, N = 10^4$, particle-cluster takes 15.58 s while cluster-particle takes 1.60 s, so cluster-particle is faster when the targets outnumber the sources. For $M = N$, particle-cluster is about twice as fast as cluster-particle. Similar trends are seen in Figure 4 for other values of the order p.

Table 1: Test case 1, CPU time (s) with M targets and N sources, $(M, N) \in \{10^4, 10^5, 10^6\}$, (a) direct sum, (b,c) treecodes with order $p = 4$, MAC parameter $\theta = 0.75$, yielding error $E \approx 5 \cdot 10^{-3}$.

	$M = 10^4$	$M = 10^5$	$M = 10^6$
(a) direct sum			
$N = 10^4$	1.01	10.10	101.13
$N = 10^5$	10.00	100.38	1015.22
$N = 10^6$	100.59	1004.65	10325.48
(b) particle-cluster			
$N = 10^4$	0.16	1.55	15.58
$N = 10^5$	0.37	3.10	30.47
$N = 10^6$	1.29	5.81	51.11
(c) cluster-particle			
$N = 10^4$	0.35	0.67	1.60
$N = 10^5$	3.32	6.43	11.31
$N = 10^6$	34.00	64.60	111.74

Next we discuss how the CPU time scales with the system size M, N. In examining the treecode algorithms, we saw heuristically that the operation count is $O((M + N) \log N)$ for the particle-cluster treecode and $O((M+N) \log M)$ for the cluster-particle treecode. However the numerical results indicate different scaling for the actual CPU run time. For example, the particle-cluster CPU time in Table 1b depends almost linearly on M and sublinearly

on N, indicating that the run time scales like $O(M \log N)$; this is work done in evaluating the potentials, assuming the moments are known. Conversely, the cluster-particle CPU time in Table 1c depends almost linearly on N and sublinearly on M, indicating that the run time scales like $O(N \log M)$; this is the work done in stage 1 in the loop over the source particles. The difference between the expected operation count and the actual CPU run time scaling may be due to several factors, e.g. (1) the CPU run time is affected not only by the operation count but also by memory access and communication costs, (2) the treecodes use direct summation at the leaves of the tree and this was not accounted for in the operation count, (3) the other terms in the operation count may become more important for larger system sizes.

Effect of MAC parameter in test case 1

Next we consider the effect of the MAC parameter θ on the treecode performance. Recall that the Taylor approximation is applied when $r/R \leq \theta$. Hence smaller θ forces the code to descend deeper in the tree, where the cluster radii r are smaller, resulting in smaller error and larger CPU time. Figure 5 displays the CPU time versus error for two MAC parameters, $\theta = 0.75$ and $\theta = 0.5$, and two particle systems, (a) $M = 10^4, N = 10^6$, (b) $M = 10^6, N = 10^4$. The main features are summarized as follows.

- For a given MAC parameter θ, increasing the order p leads to smaller error and larger CPU time.
- For a given order p, decreasing the MAC parameter θ leads to smaller error and larger CPU time.
- For a given MAC parameter θ and order p, the two treecodes have similar error. In addition, the error does not depend strongly on the system size M, N.
- For a given level of error, the particle-cluster treecode has smaller CPU time in Figure 5a and the cluster-particle treecode has smaller CPU time in Figure 5b. This agrees with the previous finding that particle-cluster is faster when the sources outnumber the targets, and cluster-particle is faster when the targets outnumber the sources.

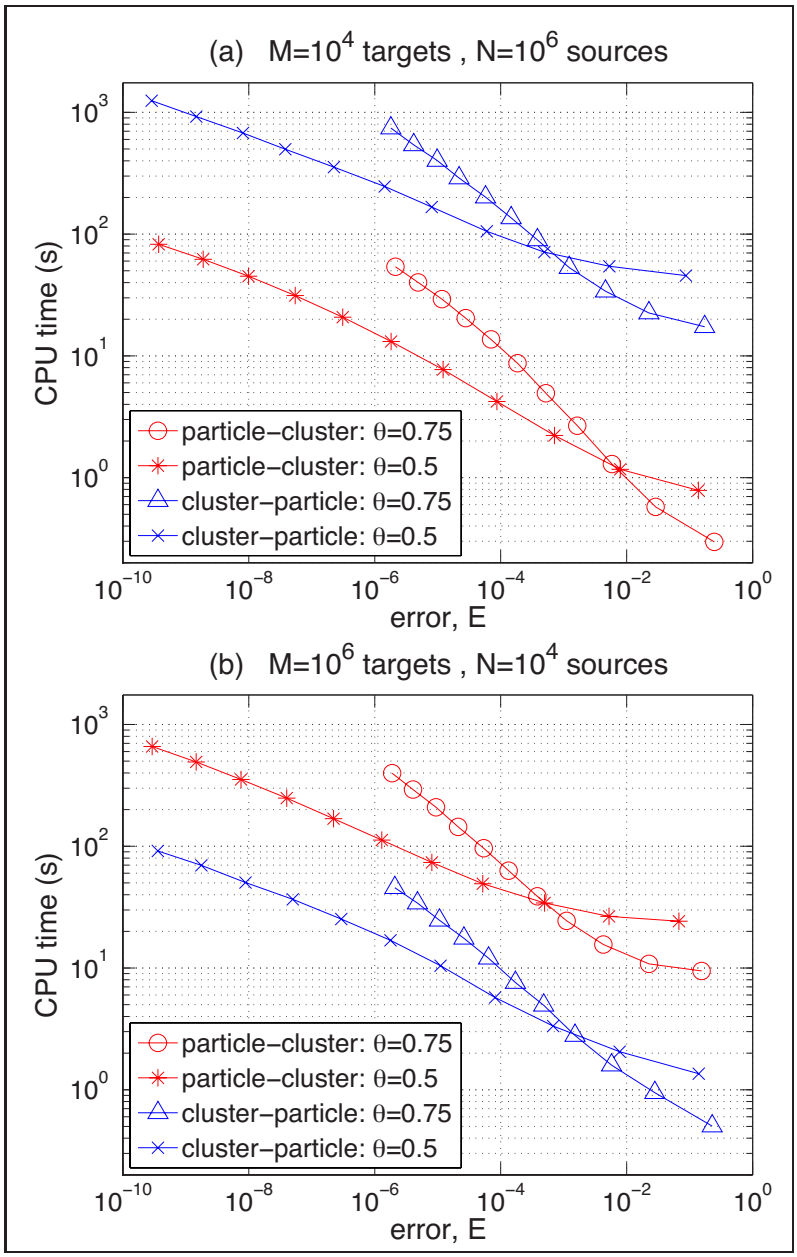


Figure 5: Test case 1, treecode CPU time versus error, order $p = 2k, k = 0, \ldots, 10$ (increasing from right to left on each line), MAC parameter $\theta = 0.75$ ($\bigcirc, \triangle$), $\theta = 0.5$ ($*, \mathsf{x}$), M targets and N sources, (a) $M = 10^4, N = 10^6$, particle-cluster is faster, (b) $M = 10^6, N = 10^4$, cluster-particle is faster.

- There is a crossover in CPU time near error $E \approx 10^{-2} - 10^{-3}$. To obtain error larger than the crossover, it is more efficient to use the larger MAC parameter $\theta = 0.75$. Conversely, to obtain error smaller than the crossover, it is more efficient to use the smaller MAC parameter $\theta = 0.5$.

- From Table 1 we see that the direct sum CPU time for these systems is about 100 s, so depending on the required level of accuracy, the treecode CPU time can be substantially smaller.

Treecode performance in test case 2

Test case 2 is concerned with particle-mesh computations in a cube. Table 2 displays the error, CPU time, and memory usage for two systems, (a) $M = 32^3 = 32768$ targets on a grid, $N = 10^6$ random sources, and (b) $M = 10^6$ random targets, $N = 32^3 = 32768$ sources on a grid. The order is $p = 4, 8, 12$ and the MAC parameter is $\theta = 0.75$.

Examining first the error in Table 2, we see that (1) the error decreases as the order p increases, (2) the two treecodes have comparable error for given order p, and (3) the errors for the two systems are comparable. For example in system (a), order $p = 4$ yields error $E = 0.575 \cdot 10^{-2}$, and order $p = 12$ yields error $E = 0.693 \cdot 10^{-4}$, about two orders of magnitude smaller. System (b) has slightly smaller errors than system (a).

Next considering the CPU time in Table 2, we see that the treecodes are faster than direct summation, except for cluster-particle with order $p = 12$. As before, the relative speed of the two treecodes depends on the system size M, N.

In system (a) with $N > M$, particle-cluster is about twenty times faster than cluster-particle for a given order p. However, particle-cluster also uses more memory, although even with order $p = 12$, it uses less than four times as much memory as direct summation (117.5 MB versus 32.2 MB). The treecode memory usage depends on the number of clusters in the tree; note that particle-cluster stores the moments for each source cluster, and cluster-particle stores the power series coefficients for each target cluster. When $N > M$, the source tree has more levels than the target tree, and this explains why particle-cluster uses more memory than cluster-particle.

Table 2: Test case 2, treecode performance, two systems (a,b) representing particle-mesh computations, order $p = 4, 8, 12$, MAC parameter $\theta = 0.75$.

(a) $M = 32^3$ targets on a grid, $N = 10^6$ random sources				
	method	$p = 4$	$p = 8$	$p = 12$
error, E	particle-cluster	0.575e-2	0.519e-3	0.693e-4
	cluster-particle	0.548e-2	0.490e-03	0.764e-4
CPU time (s)	particle-cluster	2.4	7.9	21.2
	cluster-particle	47.2	169.7	411.9
	direct sum	332.8		
memory (MB)	particle-cluster	39.6	64.7	117.5
	cluster-particle	33.3	36.1	42.7
	direct sum	32.2		

(b) $M = 10^6$ random targets, $N = 32^3$ sources on a grid				
	method	$p = 4$	$p = 8$	$p = 12$
error, E	particle-cluster	0.509e-2	0.419e-3	0.588e-4
	cluster-particle	0.478e-2	0.402e-3	0.522e-4
CPU time (s)	particle-cluster	21.3	72.0	197.9
	cluster-particle	3.8	11.9	28.1
	direct sum	330.2		
memory (MB)	particle-cluster	40.6	43.5	50.1
	cluster-particle	49.2	72.1	124.9
	direct sum	32.2		

In system (b) with $M > N$, cluster-particle is about six times faster than particle-cluster for a given order p. As before, the faster algorithm uses more memory due to the larger number of clusters in the tree, although even with order $p = 12$, cluster-particle uses less than four times as much memory as direct summation (124.9 MB versus 32.2 MB).

CONCLUSIONS

We compared two alternative treecodes for computing electrostatic potentials in charged particle systems with M targets and N disjoint sources. The particle-cluster treecode partitions the sources into an octree and applies a far-field approximation[11,19,24], while the cluster-particle treecode instead partitions the targets into an octree and applies a near-field approximation[25]. The cluster-particle treecode was developed recently in the context of radial basis function approximations[25], while the present work concerns the application to electrostatics, especially for $M \neq N$.

We documented the accuracy, CPU run time, and memory usage of the treecodes, showing how their performance depends on the system size M, N, the approximation order p, and the MAC parameter θ. To keep the discussion simple, we chose representative values for the maximum number of particles in a leaf, M_0, N_0.

Depending on the required error, the treecodes run at a fraction of the time taken by direct summation, and the treecode memory usage is a relatively modest factor more than the direct sum memory usage. The particle-cluster treecode is faster for $N > M$, i.e. when the sources outnumber the targets, and the cluster-particle treecode is faster for $M > N$, i.e. when the targets outnumber the sources. In addition, the code can be optimized with respect to either CPU time or memory usage by tuning the treecode parameters (order p, MAC parameter θ). Hence the two treecodes provide useful tools for computing electrostatic potentials in charged particle systems with disjoint targets and sources.

We mentioned several possible applications of the treecodes for systems with disjoint targets and sources, e.g. particle-mesh computations and protein docking. In this work we focused on computing the electrostatic potential, but the treecodes can be readily extended to compute the gradient of the potential[19,24] and hence the electrostatic force. The treecodes

can also be applied to other kernels for which the necessary recurrence relations are known including the real space Ewald potential[19], power law potential[20], and the screened Coulomb potential[21]. A particle-cluster treecode has been implemented for the generalized Born potential[22], although as yet only for low order p.

Before concluding we comment briefly on parallelization of treecodes[30–32]. In the particle-cluster treecode, the target computations are independent of each other (Algorithm 1, line 5), so the work can be done concurrently on multiple processors. In this case, if the available memory on each processor is large enough to hold the entire source tree, then a replicated data approach can be used[33,34]. Otherwise a distributed memory approach can be used, for example as in large-scale gravitational simulations of interacting point masses[30,32,35]. Several approaches have been developed for load balancing in parallel treecode simuations[30–32,36–38]. In the cluster-particle treecode, the source computations in stage 1 are independent of each other (Algorithm 2, line 5), so this portion of the work can also be done concurrently. However a comparison of the parallel performance of the two treecodes is reserved for future investigation.

ACKNOWLEDGEMENTS

This work was supported by NSF grants DMS-085487 and DMS-0915057. We thank Hans Johnston for making his Fortran90 particle-cluster treecode available.

References

1. M. P. Allen, D. J. Tildesley, Computer Simulation of Liquids, Oxford University Press, **1987**.

2. T. Schlick, Molecular Modeling and Simulation. An Interdisciplinary Guide, Springer, **2002**.

3. R. W. Hockney, J. W. Eastwood, Computer Simulation Using Particles, McGraw-Hill, **1981**.

4. T. Darden, D. York, L. Pedersen, *J. Chem. Phys.* **1993**, *98*, 10089-10092.

5. U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, L. G. Pedersen, *J. Chem. Phys.* **1995**, *103*, 8577-8593.

6. D. S. Cerutti, R. E. Duke, T. A. Darden, T. P. Lybrand, *J. Chem. Theory Comput.* **2009**, *5*, 2322-2338.

7. A. Brandt, A. A. Lubrecht, *J. Comput. Phys.* **1990**, *90*, 348-370.

8. B. Sandak, *J. Comput. Chem.* **2001**, *22*, 717-731.

9. R. D. Skeel, I. Tezcan, D. J. Hardy, *J. Comput. Chem.* **2002**, *23*, 673-684.

10. J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, K. Schulten, *J. Comput. Chem.* **2007**, *28*, 2618-2640.

11. J. Barnes, P. Hut, *Nature* **1986**, *324*, 446-449.

12. L. Greengard, V. Rokhlin, *J. Comput. Phys.* **1987**, *73*, 325-348.

13. J. Carrier, L. Greengard, V. Rokhlin, *SIAM J. Sci. Stat. Comput.* **1988**, *9*, 669-686.

14. H. Cheng, L. Greengard, V. Rokhlin, *J. Comput. Phys.* **1999**, *155*, 468-498.

15. M. R. Pincus, H. A. Scheraga, *J. Phys. Chem.* **1977**, *81*, 1579-1583.

16. I. Klapper, R. Hagstrom, R. Fine, K. Sharp, B. Honig, *Proteins* **1986**, *1*, 47-59.

17. N. A. Baker, D. Sept, S. Joseph, M. J. Holst, J. A. McCammon, *Proc. Nat. Acad. Sci. USA*, **2001**, *98*, 10037–10041.

18. J. Wang, R. Luo, *J. Comput. Chem.* **2010**, *31*, 1689-1698.

19. Z.-H. Duan, R. Krasny, *J. Chem. Phys.* **2000**, *113*, 3492-3495.

20. Z.-H. Duan, R. Krasny, *J. Comput. Chem.* **2001**, *22*, 184-195.

21. P. Li, H. Johnston, R. Krasny, *J. Comput. Phys.* **2009**, *228*, 3858-3868.

22. Z. Xu, *Phys. Rev. E* **2010**, *81*, 020902(R).

23. J. H. Strickland, R. S. Baty, *J. Comput. Phys.* **1998**, *142*, 123-128.

24. K. Lindsay, R. Krasny, *J. Comput. Phys.* **2001**, *172*, 879-907.

25. Q. Deng, T. A. Driscoll, *SIAM J. Sci. Comput.* **2012**, *34*, A1126-A1140.

26. J. K. Salmon, M. S. Warren, *J. Comput. Phys.* **1994**, *111*, 136-155.

27. H. Johnston, `www.math.umass.edu/~johnston/newtreecode.html`

28. H. Boateng, `sourceforge.net/projects/compare-pc-cp`

29. G. Dahlquist, Å. Björk, Numerical Methods in Scientific Computing, SIAM, **2008**.

30. M. S. Warren, J. K. Salmon, *Comput. Phys. Commun.* **1995**, *87*, 266-290.

31. J. P. Singh, C. Holt, T. Totsuka, A. Gupta, J. Hennessy, *J. Parallel Distr. Comput.* **1995**, *27*, 118-141.

32. J. Dubinski, *New Astronomy* **1996**, *1*, 133-147.

33. D. Liu, Z.-H. Duan, R. Krasny, J. Zhu, Proc. of *18th International Parallel and Distributed Processing Symposium* **2004**, IEEE Computer Society Press.

34. W.H. Geng, R. Krasny, *J. Comput. Phys.* **2013**, *247*, 62-78.

35. M. Winkel, R. Speck, H. Hübner, L. Arnold, R. Krause, P. Gibbon, *Comput. Phys. Commun.* **2012**, *183*, 880-889.

36. A. Grama, V. Kumar, A. Sameh, *Parallel Comput.* **1998**, *24*, 797-822.

37. Y. M. Marzouk, A. F. Ghoniem, *J. Comput. Phys.* **2005**, *207*, 493-528.

38. H. Feng, A. Barua, S. Li, X. Li, *Commun. Comput. Phys.*, to appear.