Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Computational Physics 200 (2004) 192–210
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Variable order revised binary treecode

Qian Xi Wang

Maritime Research Centre, Nanyang Technological University, 50 Nanyang Avenue, Singapore 630798, Singapore

Received 30 December 2003; received in revised form 29 March 2004; accepted 5 April 2004
Available online 7 May 2004

Abstract

Three essential improvements are described to the treecode in terms of the expansion formula, the choice of the expansion order as well as the tree structure. Firstly, the multipole expansion is based on the real spherical harmonic functions to reduce the CPU time. Secondly, the expansion order is given in terms of the ratio of the distance of a field point to a source box to the box size, which reflects the relative error of the expansion. With that, a large portion of the sources has been evaluated by the multipole expansion at low levels of the source tree, which is around two-thirds of sources at the first two levels of the tree averagely. The algorithm reduces the CPU time dependency on expansion order p from $O(p^2)$ of the classical treecode to be lower than a linear dependency in p_{max}, where p_{max} is the maximum expansion order used in the variable order expansion. Thirdly, a revised binary tree is built by performing the bisections thrice at each tree level, discarding the boxes generated in the first two bisections and remaining only the boxes generated in the last one. This tree avoids the disadvantage of a binary treecode demanding significantly more CPU time than an oct-treecode. It has high adaptiveness to the source distribution and perfect load balancing for performing the parallelization. Simulations are carried out for N vortex elements and N field points distributed randomly in a cube, a 5:1:1 parallelepiped, and a 10:1:1 parallelepiped, using the oct-tree and revised binary tree, respectively. The algorithm is an order of magnitude faster than those of Strickland et al. [ESAIM: Proceedings 7 (1999) 408], Warren and Salmon [Comput. Phys. Commun. 87 (1995) 266], and Lindsay and Krasny [J. Comput. Phys. 172 (2001) 879]. Simulations also demonstrate the efficiency of the revised binary treecode for an inhomogeneous source distribution.

Keywords: Treecode; Fast adaptive algorithm; Multipole expansion; Revised binary tree; Vortex method

1. Introduction

The classical N-body problem simulates the movement of a set of particles under the influence of the gravitational, electrostatic, or other typed forces. N-body algorithms have a number of important applications in the fields of astrophysics, plasma physics, molecule dynamics, fluid dynamics, computer graphics, and linear matrix equations, etc [4,5]. The problem can be simulated straightforwardly by computing all N^2 pairwise interactions, but it is prohibitively expensive for a large-scale system. The treecode by Barnes and

E-mail address: cqxwang@ntu.edu.sg (Q.X. Wang).

0021-9991/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2004.04.003

Hut [4] and the fast multipole expansion method (FMM) by Greengard and Rokhlin [5] dramatically accelerate this kind evaluation to $O(N \log N)$ and $O(N)$, respectively, with controllable errors. The basic idea of the treecode is to approximate the potential at a distant target due to a source cluster by an expansion in the spherical harmonic functions, termed as the multipole expansion. FMM goes one step further by converting a multipole expansion to a local expansion to represent the target value in a local domain [5–8].

As compare to FMM, the treecode is simple to be implemented, requiring less memory and having higher parallel efficiency [9,10]. The treecode was implemented to molecular dynamics [11–16], and astrophysics [17–20], to solve the dense linear matrix equations [21–23], among others. It was developed and applied in the vortex method for fluid flows [24–34]. However, the treecode does not perform well at high level accuracy. Blelloch and Narlikar [9] made a direct comparison between their treecode and FMM code for the potential problem of random sources. They found that, at RMS Error $= O(10^{-3})$, FMM did not outperform the treecode until $N > 10^8$; at RMS Error $= O(10^{-5})$, the FMM code was faster than the treecode for $N > 10^4$. For $N = 7 \times 10^6$ point sources and N field points at RMS-error $= 2 \times 10^{-3}$, the operation count of the treecode was about one third that of the FMM code. At RMS-error $= 2 \times 10^{-5}$, the operation count of the treecode was about twice that of the FMM code.

The conventional treecodes use a uniform expansion order. Most treecodes use only three terms up to the quadruple moment. To achieve higher accuracy, one needs to use a higher order expansion with its CPU time increasing at $O(p^2)$. Alternatively, one can select a larger minimum well-separation distance for employing the multipole expansion; consequently, the costly direct evaluation has to be performed for more sources.

The variable order expansion technique was implemented to speedup the treecode at high accuracy by Petersen et al. [13]. A higher expansion order is employed for source boxes near a field leaf, and gradually lower expansion order for source boxes far away. Salmon and Warren [17] derived an error-bound of the multipole expansion for vortex elements. Grama et al. [22,23] improved their treecode accuracy at a small computational overhead by increasing the expansion order with the strength sum of a source cluster, based on the error bound of the multipole expansion for potentials. The error bounds were derived assuming a field point is far away from a source cluster, but the expansion is usually performed as a field point is not far away. It has been found that the error-bounds are generally over strict. For this reason, Lindsay and Krasny [3] used the first term of the expansion as a heuristic estimate of the error for their treecode.

In the above works, the expansion order is determined according to the error produced by the multipole expansion, which is proportional to the strength sum of a source box. Consequently, the multipole expansion is apt to be performed on small boxes at high levels of the source tree. As an alternative, the expansion order in this work is given in terms of the ratio of the distance of a field point to a source box to the box size. Simulations in Section 4 showed the ratio reflects the relative error of the multipole expansion. With that, the multipole expansion in the treecode is performed on large boxes at low levels of the source tree. In addition, the relative error of the treecode is limited when the relative errors of all of the multipole expansions performed are small.

The other main part of a treecode beside the multipole expansion is the tree structure, which is a hierarchical partition of the computation domain. Most of treecodes are based on the oct-tree, staring with a cube containing all particles, and level $L + 1$ of the tree being obtained by subdividing all the cubes in level L into eight equal smaller cubes. Its adaptiveness to the source distribution is low. Clarke and Tutty [34] and Strickland et al. [1] implemented a binary tree, in which, a box is divided along its longest dimension into two subdomains containing equal particles, termed as the bisection. The binary treecode has high adaptiveness to the source distribution and high parallel efficiency. However, at the same resolution, a binary tree has about 1.75 times more boxes and three times more levels than an oct-tree demanding significantly more CPU time [1].

A revised binary tree is proposed as follows to keep the advantages of an oct-tree and a binary tree and avoid their disadvantages. At each tree level, the bisection is performed for three times in a row, discarding

the boxes generated in the first two bisections and remaining only the boxes generated in the last one. By doing this, a box in this tree has eight children as that in an oct-tree. In the meantime, the revised binary treecode has high adaptiveness to the source distribution and high parallel efficiency.

The rest of the paper is organized as follows. Firstly, the multipole expansion is introduced based on the real spherical harmonic functions in Section 2. A revised binary tree is illustrated in Section 3. A variable order multipole expansion and the criterion for choosing the expansion order are described in Section 4. In Section 5, the treecode is first compared with those of Strickland et al. [1], Warren and Salmon [2], and Lindsay and Krasny [3]. Simulations are then described for N vortex elements and N field points distributed randomly in a cube, a 5:1:1 parallelepiped, and a 10:1:1 parallelepiped, using the oct-treecode and revised binary treecode, respectively. Section 6 contains the summary and conclusions of the present work.

2. Multipole expansion in real spherical harmonic functions

Consider N point vortexes of circulations $\vec{\omega}_1, \vec{\omega}_2, \ldots, \vec{\omega}_N$ located at $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_N$ with spherical coordinates (r_j, α_j, β_j), $j = 1, 2, \ldots, N$. The velocity at field point $\boldsymbol{p}$ with spherical coordinates (ρ, θ, ψ) induced by the N vortex points is given by the Biot–Savart law

$$\boldsymbol{v}(\boldsymbol{p}) = \frac{1}{4\pi}\sum_{j=1}^{N}\frac{\vec{\omega}_j \times (\boldsymbol{p}-\boldsymbol{q}_j)}{|\boldsymbol{p}-\boldsymbol{q}_j|^3} = \frac{1}{4\pi}\sum_{j=1}^{N}\vec{\omega}_j \times \nabla_{\boldsymbol{q}}\left(\frac{1}{|\boldsymbol{p}-\boldsymbol{q}_j|}\right). \tag{1}$$

Without losing generality, let points $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_N$ located inside a sphere of radius a centered at the origin. Suppose field point $\boldsymbol{p}$ is outside of the sphere, i.e., $|\boldsymbol{p}| > a > |\boldsymbol{q}_j|$, $1/|\boldsymbol{p}-\boldsymbol{q}_j|$ can be expanded in terms of the spherical harmonic functions as follows:

$$\frac{1}{|\boldsymbol{p}-\boldsymbol{q}_j|} = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} r_j^n Y_n^{-m}(\alpha_j, \beta_j)\frac{Y_n^m(\theta,\psi)}{\rho^{n+1}}, \tag{2}$$

where spherical harmonic function $Y_n^m(\theta,\psi)$ is

$$Y_n^m(\theta,\psi) = \sqrt{\frac{(n-|m|)!}{(n+|m|)!}}P_n^{|m|}(\cos\theta)\mathrm{e}^{\mathrm{i}m\psi} \quad \text{for } n = 1 \text{ to } \infty \text{ and } m = -n \text{ to } n \tag{3}$$

with $P_n^m(x)$ being the associate Legendre function.

Each term in (2) consists of two parts of $r_j^n Y_n^{-m}(\alpha_j, \beta_j)$ and $Y_n^m(\theta,\psi)/\rho^{n+1}$, in terms of information associated only with the sources and field point, respectively. Noticing this and inserting (2) into (1), one obtains

$$\boldsymbol{v}(\boldsymbol{p}) = \sum_{n=1}^{\infty}\sum_{m=-n}^{n}\boldsymbol{M}_n^m\frac{Y_n^m(\theta,\psi)}{\rho^{n+1}} \approx \sum_{n=1}^{p}\sum_{m=-n}^{n}\boldsymbol{M}_n^m\frac{Y_n^m(\theta,\psi)}{\rho^{n+1}}, \tag{4}$$

where P is the expansion order and $\boldsymbol{M}_n^m$ are the multipole expansion coefficients given by source parameters only

$$\boldsymbol{M}_n^m = \sum_{j=1}^{N}\vec{\omega}_j \times \nabla\left(r_j^n Y_n^{-m}(\alpha_j, \beta_j)\right). \tag{5}$$

Thus, the target values at various field points can be calculated by first accumulating the source influences into the multipole expansion coefficients of (5) and then evaluating the single truncated multipole expansion of (4).

The derivative in (1) may be performed either to source coordinate $\boldsymbol{q}$ or to field coordinate $\boldsymbol{p}$. We have purposely chosen to perform the derivative to source coordinate $\boldsymbol{q}$. With that, the derivative calculation is performed while calculating the multipole expansion coefficients of (5). Otherwise, if we performed the derivative to field coordinate $\boldsymbol{p}$ in (1) and calculated the derivative at the function evaluation stage of (4), substantially more CPU time would be needed, since this step is the bottleneck of a treecode.

The multipole expansion of (4) can be recast in terms of the real spherical harmonic functions as follows:

$$\boldsymbol{v}(p) = \sum_{n=1}^{\infty}\sum_{m=0}^{n}\frac{\boldsymbol{C}_n^m Y_{nm}^c(\theta,\psi) + \boldsymbol{D}_n^m Y_{nm}^s(\theta,\psi)}{\rho^{n+1}} \approx \sum_{n=1}^{p}\sum_{m=0}^{n}\frac{\boldsymbol{C}_n^m Y_{nm}^c(\theta,\psi) + \boldsymbol{D}_n^m Y_{nm}^s(\theta,\psi)}{\rho^{n+1}}, \tag{6}$$

where $Y_{nm}^c(\theta,\psi)$ and $Y_{nm}^s(\theta,\psi)$ are the real spherical harmonic functions

$$Y_{nm}^c(\theta,\psi) = P_n^m(\cos\theta)\cos m\psi, \quad Y_{nm}^s(\theta,\psi) = P_n^m(\cos\theta)\sin m\psi \quad \text{for } n = 1 \text{ to } p \text{ and } m = 0 \text{ to } n. \tag{7}$$

$\boldsymbol{C}_n^m$ and $\boldsymbol{D}_n^m$ are the multipole expansion coefficients determined by the source parameters

$$\boldsymbol{C}_n^m = \begin{cases} 2\frac{(n-m)!}{(n+m)!}\sum_{j=1}^{N}\vec{\omega}_j \times \boldsymbol{C} & \text{for } n = 0, \\ \frac{(n-m)!}{(n+m)!}\sum_{j=1}^{N}\vec{\omega}_j \times \left(\boldsymbol{C}\cos\left(m\beta_j\right) + \boldsymbol{D}\sin(m\beta_j)\right) & \text{for } n > 0, \end{cases} \tag{8a}$$

$$\boldsymbol{D}_n^m = \frac{(n-m)!}{(n+m)!}\sum_{j=1}^{N}\vec{\omega}_j \times \left(\boldsymbol{C}\sin(m\beta_j) - \boldsymbol{D}\cos(m\beta_j)\right) \tag{8b}$$

with

$$\boldsymbol{C} = 2\frac{\sin\alpha_j}{r_j^{n-1}}\left(n\sin\alpha_j P_n^m\left(\cos\alpha_j\right)\boldsymbol{e}_r + \left(n\cos\alpha_j P_n^m\left(\cos\alpha_j\right) - (n+m)P_{n-1}^m\left(\cos\alpha_j\right)\right)\boldsymbol{e}_\alpha\right), \tag{9a}$$

$$\boldsymbol{D} = -2\frac{\sin\alpha_j}{r_j^{n-1}} m P_n^m(\cos\alpha)\boldsymbol{e}_\beta, \tag{9b}$$

where $\boldsymbol{e}_r$, $\boldsymbol{e}_\alpha$ and $\boldsymbol{e}_\beta$ are the three unit coordinate vectors for the spherical coordinate system.

The multipole expansion of (6) based on the real spherical harmonic functions reduces the CPU time and memory requirements significantly. To elaborate this comment, let us compare the two approaches. To calculate velocities at N field points, the function evaluation of (4) or (6) requires $O(p^2N)$ operation. In principle the calculation of the multipole expansion coefficients of (5) or (8) requires $O(p^2N)$ operation too. However, this step costs only a small part of the CPU time. The simulations in Section 5 revealed that the CPU time is consumed predominantly for the function evaluation, being from 93% to 97% for 10^4–10^6 vortex elements distributed randomly in a cube.

The function evaluation consists of two steps: calculating the spherical harmonic functions needed in (4) or (6), and evaluating (4) or (6). Denoting $O_c(f)$ as the operation count of f, and comparing (3) and (7), one has

$$\frac{O_c(Y_n^m(\theta,\psi))}{O_c(Y_{nm}^c(\theta,\psi)) + O_c(Y_{nm}^s(\theta,\psi))} > 1. \tag{10}$$

Considering the number of the spherical functions needed in the two approaches, one can see that the operation count of $Y_n^m(\theta,\psi)$ (for $n = 1$ to p and $m = -n$ to n) needed in (4) is more than $2(p+2)/(p+3)$ times that of the real spherical harmonic functions $Y_{nm}^c(\theta,\psi)$ and $Y_{nm}^s(\theta,\psi)$ (for $n = 1$ to p and $m = 0$ to n) needed in (6). With the spherical harmonic functions calculated, the operation count of (4) is

$4(p+2)/(p+3)$ times that of (6). Therefore, the multipole expansion of (6) reduces the CPU time significantly.

We further compare the memory requirements of the two approaches. The multipole expansion coefficients of source boxes consist of a substantial part of the total memory requirement. The memory requirement of real vectors $\boldsymbol{C}_n^m$ and $\boldsymbol{D}_n^m$ (for $n = 1$ to p and $m = 0$ to n) is only about $(p+1)/(2p+1)$ times that of complex vector $\boldsymbol{M}_n^m$ (for $n = 1$ to p and $m = -n$ to n). Each coefficient of $\boldsymbol{M}_n^m$ is a complex vector, and it needs 24 bytes in single precision. The memory requirement of $\boldsymbol{M}_n^m$ (for $n = 1$ to p and $m = -n$ to n) for all source box is of

$$M = 24p(2p+1)N_{\text{otc}} \text{ (Bytes)},$$

where N_{otc} is the number of source boxes. For a uniform source distribution in a cube, $N_{\text{oct}} = (8^{l+1} - 1)/7$, where l is the level of the tree. As an illustration, $M \approx 560$ MB at $p = 6$ and $l = 6$. For this case, about 260 MB memory is saved with the multipole expansion of (6) based on the real spherical harmonic functions.

A vortex element with the Gaussian smoothed core function, termed as the vorton, is usually deployed to provide smooth and stable flow solutions in the vortex method. Velocity $\boldsymbol{v}(\boldsymbol{p})$ at field point $\boldsymbol{p}$ generated by a vorton with circulation of $\vec{\omega}$ centered at position $\boldsymbol{q}$ is [27]

$$\boldsymbol{v}(\boldsymbol{p}) = \frac{1}{4\pi} \frac{\vec{\omega} \times (\boldsymbol{p} - \boldsymbol{q})}{|\boldsymbol{p} - \boldsymbol{q}|^3} \left[1 - \mathrm{e}^{-\left(\frac{|\boldsymbol{p}-\boldsymbol{q}|}{\delta}\right)^3} \right], \tag{11}$$

where δ is termed as the vorton radius, which was set to be 1.5 times of the average neighboring distances of vortons in the simulations in Section 5. The vorton contribution (11) is non-Coulomb interaction. With the Gaussian smoothing function decaying exponentially, (11) can be approximated as a point vortex with a relative error from 3.4×10^{-4} to 1.6×10^{-7} when a cutoff distance is chosen from 2δ to 2.5δ from the vorton centre. The vorton, in this work, has been approximated as a point vortex with a relative error at 1.6×10^{-7} with a cutoff distance at 2.5δ from the vorton centre. As such, significantly more direct calculations have to be performed and the speedup of a treecode or FMM code is slow down significantly as compared to pure Coulomb interactions.

3. Revised binary tree structure

The multipole expansion can only be performed when the sources and targets are well separated. To satisfy this condition, the sources and targets must be clustered properly so that the multipole expansions can be performed efficiently without significant loss in accuracy. This is achieved by the use of a hierarchical partition of the computation domain, termed as the tree structure. Most treecodes are based on the oct-tree, staring with a cube containing all particles termed as the root. Level $L+1$ of the tree is obtained by subdividing all the cubes in level L into eight equal smaller cubes. The end boxes in the tree are termed as the leaves. In an adaptive approach, only boxes with sufficient particles are further divided, leading to an irregular tree. A non-adaptive tree is a complete tree whose leaves all have the same depth, which is not suitable for non-uniform particle distributions. In this work, two adaptive oct-trees are generated for sources and targets, respectively, for flexibility, since they may be different in applications.

Strickland et al. [1] implemented a binary tree for their treecode, which yielded a very high parallel efficiency. In fact, they obtained over 90% parallel efficiency for up to 2048 processors. The binary tree was first introduced by Clarke and Tutty [34] for their two-dimensional treecode. The root of the binary tree is defined by the range of sources in the x, y and z directions. This box is then divided into two sub-domains along its longest dimension with equal numbers of sources. Each sub-domain is then shrunk to the smallest box to exactly contain its designated sources. This bisection and shrinkage continue until the number of

sources at the finest level is below some specified value. A binary tree is adaptive to the source distribution and its parallel efficiency is very high.

At the same resolution, the depth of a binary tree is three times that of an oct-tree. The box number of a binary-tree of depth $3l$ is $N_{\text{bn}} = 2^{3l+1} - 1$, and the box number of an oct-tree of depth l is $N_{\text{oct}} = (8^{l+1} - 1)/7$. Thus

$$N_{\text{bn}}/N_{\text{oct}} = 7\left(2^{3l+1} - 1\right)/\left(8^{l+1} - 1\right) \approx 1.75 \quad \text{as } l \geqslant 3.$$

The binary tree has about 1.75 times more boxes than an oct-tree at the same resolution. For sources evenly distributed in a cube, the operation count of the oct-treecode C_{cub} and that of the binary treecode C_{bn} are

$$C_{\text{cub}} = 189p^2N\log_8 N + 27sN, \tag{12a}$$

$$C_{\text{bn}} = 189p^2N\log_2 N + 27sN. \tag{12b}$$

The CPU time of the binary treecode is thus about three times slower than an oct-treecode.

As an alternative, a revised binary tree is proposed in this paper as follows. At each tree level, the bisection and shrinkage are performed for three times in a row, discarding the boxes generated in the first two bisections and remaining only the boxes generated in the last one. The revised binary tree is similar to the traditional oct-tree used in treecodes or FMM codes. A box in this tree has eight children as that in an oct-tree, and it avoids the disadvantage of a binary tree demanding significantly more CPU time. Also, the oct-tree codes do not have to use cubic boxes (cf. [16]). But, as compare to an oct-tree, the revised binary tree has the following essential advantages. Its adaptiveness to the computational domain and source distribution is as high as that of a binary tree. In this tree, the boxes at a given level contain essentially the same number of sources (within ± 1), thus has perfect load balancing for performing the parallelization. It is an adaptive tree but it is a complete tree whose leaves all have the same depth, the parallelization of a revised binary treecode is thus as simple as that of a non-adaptive approach.

The children boxes of a parent box in a revised binary tree may not be octahedral sited. As simple illustrations, Fig. 1 shows the first level of the revised binary trees for (a) an 8:1:1 parallelepiped domain and (b) a 4:2:1 parallelepiped domain with uniform sources. Each box has at most three neighboring boxes, including itself, for the 8:1:1 parallelepiped domain (Fig. 1(a)) and six neighboring boxes for the 4:2:1 parallelepiped domain (Fig. 1(b)); all other boxes are well separated from the box and are suitable for performing the multipole expansion calculation. If an oct-tree is used, each box has eight neighboring boxes in the first level tree for both of the two cases. The revised binary tree is thus much more accommodative to a non-cube domain as compare to an oct-tree.

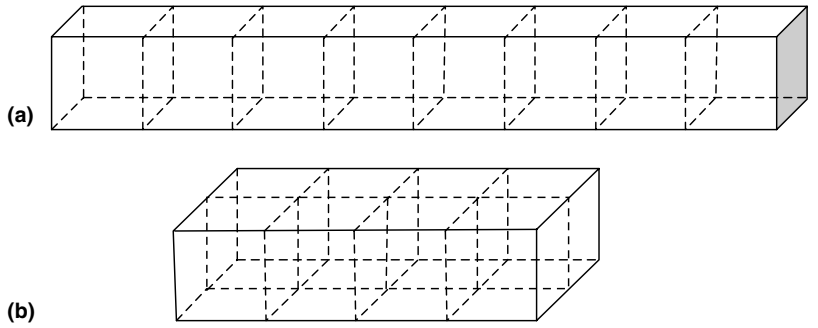

Fig. 1. The first level revised binary trees of (a) an 8:1:1 parallelepiped domain and (b) a 4:2:1 parallelepiped domain with uniform sources.

4. Variable order multipole expansion

A treecode starts by forming the multipole expansion coefficients for all source boxes. The function evaluation is then performed at field points in all field leaves (the finest boxes in the field tree). For each field leaf, starting from the coarsest level of the source tree and for all boxes at that level, a logic check and a function evaluation are conducted as follows:

(1) If the number of sources in the box is smaller than a prescribed number, which in this work is chosen at 20, the direct evaluation is used.
(2) If the number of the sources in the box is not small and the box is well separated from the field leaf, the multipole expansion evaluation is used.
(3) Otherwise, the evaluation is passed onto the next finer level at which the same logical check is performed on the boxes whose contributions have not been counted.

When the finest level of the source tree is reached, the influences from the boxes that do not meet the well-separation condition are computed using the direct method.

As discussed in Section 1, a variable order multipole expansion is implemented in this work. A uniform expansion order $p_{\max}$ is used to calculate the multipole expansion coefficients for all source boxes, since this step costs only a small part of the total CPU time. But, a variable expansion order is used to calculate the target values, the bottleneck of the treecode. A higher expansion order $p \leqslant p_{\max}$ is deployed for source boxes near a field leaf, and a gradually lower order for source boxes far away.

Ideally, the expansion order should be given in terms of the analytical error-bounds of the multipole expansion, but the available error-bounds were all obtained assuming the field point is far away from a source cluster (cf. [17]), and the multipole expansion is actually performed when the field point is not far away. It has been found that the error-bounds are generally over strict. Because of this, Lindsay and Krasny [3] used the first term of the expansion as a heuristic estimate of the error for their variable order treecode. In the previous treecodes, the expansion order is determined according to the error produced by the multipole expansion, which is proportional to the strength sum of a source box. Consequently, the multipole expansion is apt to be performed on small boxes at high levels of the source tree.

With the above considerations, the expansion order here is given in terms of separation distance D between a field leaf and a source box and diagonal length d of the source box (or the diameter of the smallest sphere containing all the sources in the box)

$$\frac{D}{d} \geqslant \theta(p), \tag{13}$$

where $\theta(p)$ depends on the accuracy required and expansion order p.

To decide $\theta(p)$, a numerical testing is devised for a field point at various distance D to a unit sphere containing various numbers of random point vortexes, with random positions and random circulations, as shown in Fig. 2(a). A single multipole expansion is used to calculate the induced velocity at the field point. Fig. 2(b) shows the error of the multipole expansion at $p = 6$ versus D/d for various numbers of point vortexes at $N = 20$, 10^2, 10^4, and 10^5. One can see that the error reduces rapidly with D/d. The error increases from three to four orders of magnitude while the number of point vortexes increases from $N = 20$ to 10^5. Therefore, for a treecode with the well-separation condition given in terms of the expansion error, the multipole expansion has to be performed on small source boxes at high levels of the source tree. This is not surprising. The more interesting feature is the variation of the relative error with N and D/d. Fig. 2(c) shows the corresponding relative error of the multipole expansion versus D/d. The relative error reduces rapidly with D/d, falling about *9 orders of magnitude* as D/d increases from 0.5 to 11. In contrast, the relative error reduces *less than one order of magnitude*, being about 5 times, as the number of the point vortexes increases in *four orders of magnitude*.

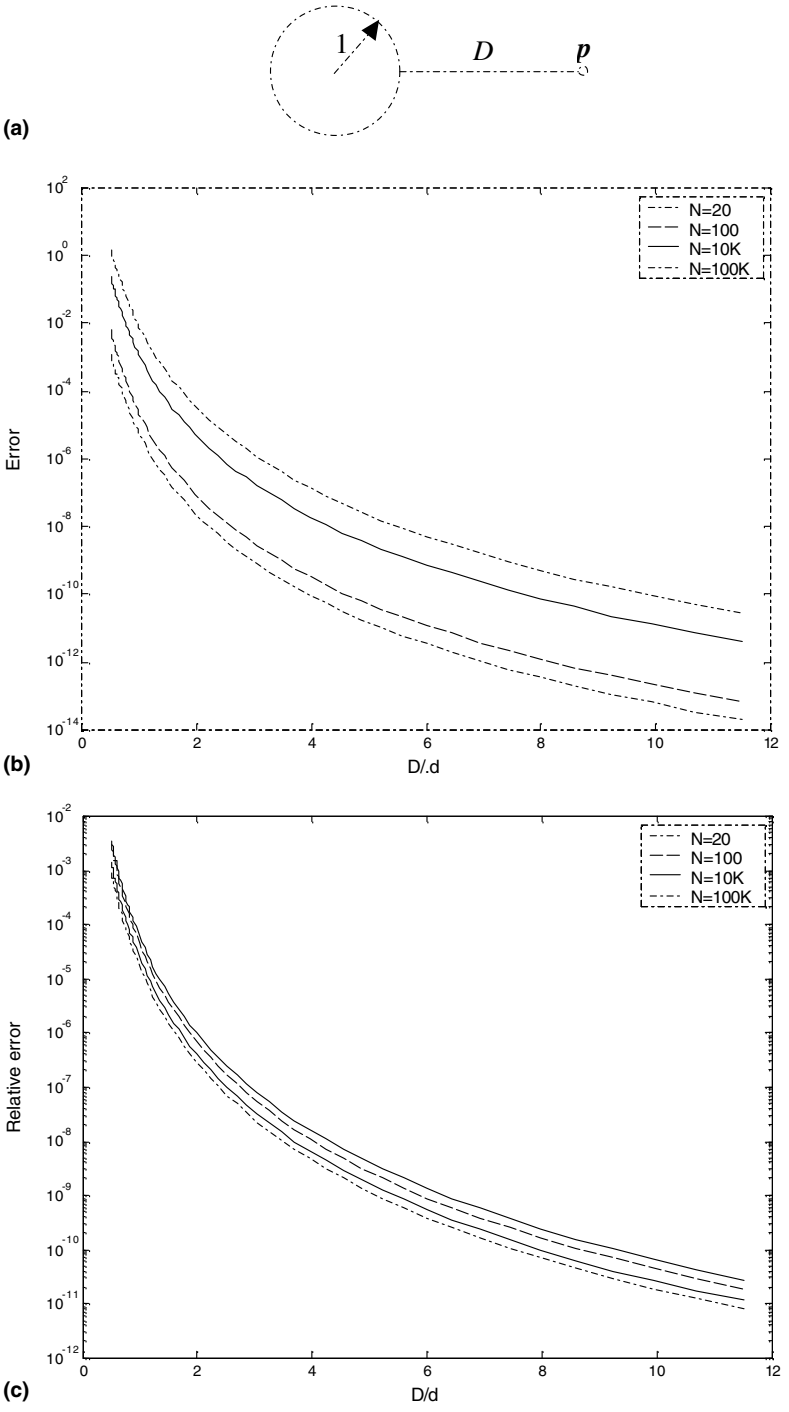

Fig. 2. (a) A unit sphere containing N random point vortexes and a field point $\boldsymbol{p}$ at distance D from the sphere. (b) The error and (c) relative error of the single multipole expansion at $p = 6$ versus D/d for $N = 20, 100, 10$ and 100 K.

With the above analyses, we reach the following conclusions. The ratio of D/d reflects the relative error of the multipole expansion and, at least, decides the order of the relative error. This feature of the multipole expansion even stands for the field point near the source cluster. The well-separation condition and the multipole expansion order thus should be based on the relative error rather than the error. With that, the multipole expansion is apt to be performed on large source boxes at the low levels of the source tree. In addition, the relative error of the treecode is limited when all of the multipole expansions are performed at a small relative error.

Fig. 3 shows the relative error of the multipole expansion versus D/d for the testing at $N = 1000$ and various expansion orders from $p = 1$ to 10. The relative error reduces rapidly with expansion order p and the ratio *of* D/d. Based on the figure, we set

$$p_{\max} = 6 \text{ and } \theta(6) = 0.75, \quad \theta(5) = 1.0, \quad \theta(4) = 1.35, \tag{14a}$$

to keep the relative error for velocity at $O(10^{-4})$; and set

$$p_{\max} = 7 \text{ and } \theta(7) = 0.75, \quad \theta(6) = 1.0, \quad \theta(5) = 1.35, \quad \theta(4) = 1.8, \tag{14b}$$

to keep the relative error for velocity at $O(10^{-5})$. Those choices of $\theta(p)$ have been evaluated in the simulations of Section 5.

5. Numerical results

The algorithm described in Sections 2–4 has been implemented in Fortran 77, and simulations have been carried out on a PC with Pentium 4 CPU 2.66 GHz and 0.99 GB of RAM. Three cases are considered for n vortons and N field points distributed randomly in a cube, a 5:1:1 parallelepiped, and a 10:1:1 parallelepiped, respectively. The three cases may represent the vortex element distributions for simulating the flow around a bluff body at different stages of the wake development.

For each case, the numerical tests are performed with both the oct-treecode and revised binary treecode, respectively. For the cube case, the results of the two treecodes were close to each other, therefore only the results of the oct-treecode will be discussed. For the two cases of 5:1:1 and 10:1:1 parallelepipeds, the speedups of the oct-treecode are just slightly better than that of the cube case, whereas the speedups of the revised binary tree-code increase significantly and will be discussed.

The velocity results and CPU times of the treecode are compared with those of the direct calculation. Obviously, it is not practical to apply the direct calculation for large-scale ensembles of field points due to excessive computation time. The direct calculation is thus only performed for $N_{\text{dir}} = N$ field points as $N \leqslant 10^5$, and for first $N_{\text{dir}} = 10^5$ field points for $N > 10^5$ and the resulting CPU time is extrapolated. RMS error E_1 and maximum relative error E_2 for velocity of the treecode are calculated at the first N_{dir} field points via the following formulae

$$E_1 = \left(\frac{\sum_{i=1}^{N_{\text{dir}}} \left| \boldsymbol{v}^{\text{tc}}(\boldsymbol{p}_i) - \boldsymbol{v}^{\text{dir}}(\boldsymbol{p}_i) \right|^2}{\sum_{i=1}^{N} \left| \boldsymbol{v}^{\text{dir}}(\boldsymbol{p}_i) \right|^2} \right)^{1/2}, \tag{15a}$$

$$E_2 = \left(\frac{\text{Max} \left| \boldsymbol{v}^{\text{tc}}(\boldsymbol{p}_i) - \boldsymbol{v}^{\text{dir}}(\boldsymbol{p}_i) \right|^2}{\text{Max} \left| \boldsymbol{v}^{\text{dir}}(\boldsymbol{p}_i) \right|^2} \right)^{1/2}, \tag{15b}$$

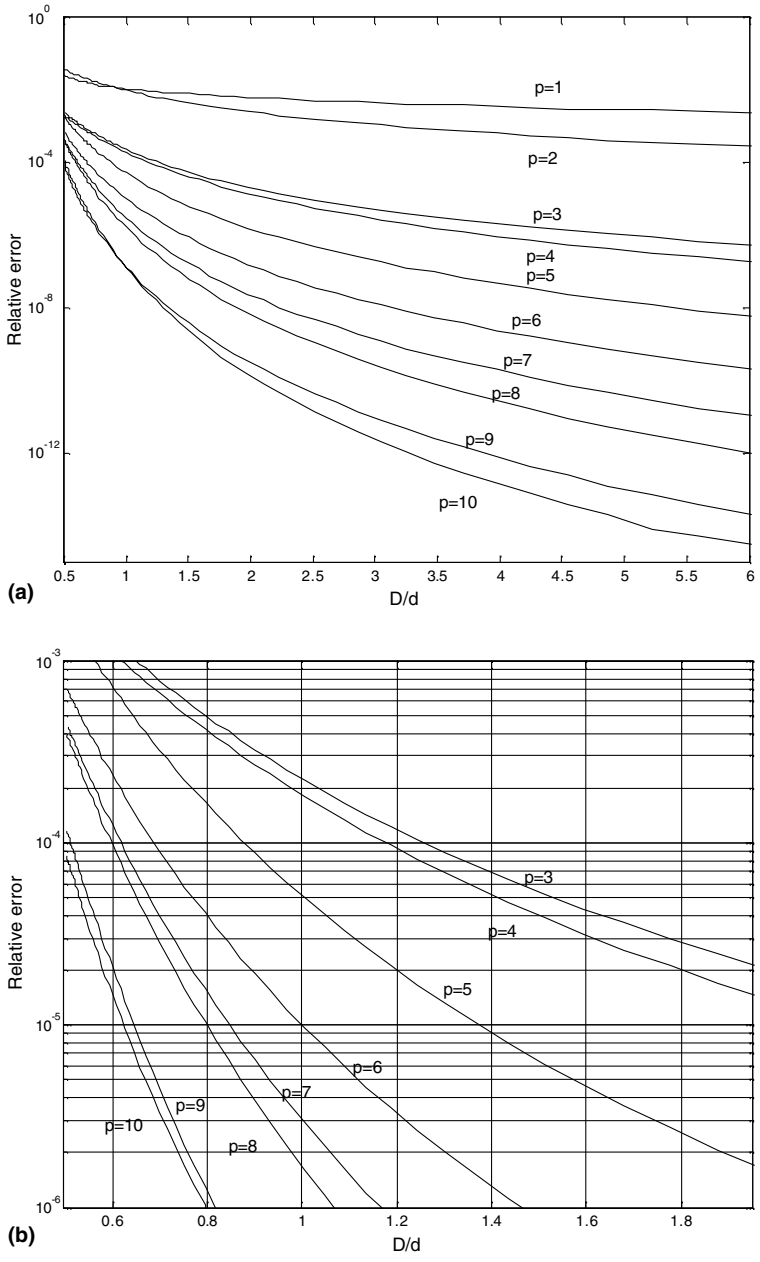

Fig. 3. (a) The relative errors of a single multipole expansion versus D/d for $N = 10^3$ point vortexes distributed randomly in a sphere at various expansion orders p. (b) A local amplified view of (a).

where $\boldsymbol{v}^{\mathrm{tc}}(\boldsymbol{p}_i)$ and $\boldsymbol{v}^{\mathrm{dir}}(\boldsymbol{p}_i)$ are the velocities at field point $\boldsymbol{p}_i$ calculated from the treecode and direct calculation, respectively. The speedup, the ratio of the CPU time of the direct calculation to that of the treecode, is used to measure the algorithm.

5.1. Cubic domain

We first consider vortons distributed randomly in a cube, a worst scenario test, since both the CPU and memory requirements of a treecode or an FMM code are at maximums for homogeneous distributions (cf. [8,25]). Table 1 lists the treecode performance in terms of the CPU time, maximum relative error and RMS error for various numbers of vortons from $N = 500$ to 3×10^6. The maximum expansion order in this case is chosen as $p_{\max} = 6$ and $\theta(p)$ in (13) is set according to (14a). With that, the RMS error and maximum relative error for velocity have been well kept at $O(10^{-4})$ as shown in the table. The breakeven point between the CPU times of the treecode and the direct calculation is less than $N = 10^3$. The treecode is of one order of magnitude faster than the direct calculation at $N = 10^4$, and is of two orders of magnitude faster at $N = 2 \times 10^5$. The speedup reaches *three orders* of magnitude at $N = 3 \times 10^6$.

Fig. 4 compares the speedup of the treecode versus the number of vortons to that of Strickland et al. [1] for the same problem and at the same level of accuracy. The speedup of the present treecode is of 60 times higher than theirs in the range from $N = 2 \times 10^4$ to 3×10^6. It should be mentioned that their treecode was mainly apt for the parallel performance. We also plotted the results of Warren and Salmon [2] for vortex elements on the surface of a sphere, and the results of Lindsay and Krasny [3] based on the Taylor expansion for vortex elements with the Rosenhead–Moore kernel on a surface. The treecode for vortex elements on a surface should be much more efficient than that for the case considered here. Nevertheless, the speedup of the present treecode is still 20 times higher than that of Warren and Salmon from $N = 10^5$ to 3×10^6, and 7 times faster than that of Lindsay and Krasny.

Fig. 5 compares the speedups versus the number of vortons N between the treecodes using real spherical harmonic functions (solid line) and the complex spherical harmonic functions (dashed line) for N vortons and N field points distributed randomly in a cube. For the later approach, $p_{\max}$ and $\theta(p)$ are chosen as same as the above, and the RMS error and maximum relative error for velocity have been kept at $O(10^{-4})$ too. Comparing the results, one can see that the approach based on the real spherical harmonic functions reduced the CPU time significantly. About 36% of the CPU time is saved at $N = 10^6$. As such, all other simulations in this paper are performed with the treecode with the real spherical harmonic functions.

5.2. Computational features of the treecode

We then proceed to analyze the computational features of the treecode. Table 2 provides the proportions of the CPU time spent for generating the source and target trees, calculating the multipole expansion coefficients for the source boxes, and evaluating the function for N vortons and N field points distributed

Table 1
Performance of the treecode at $p_{\max} = 6$ for N vortons and N field points distributed randomly in a cube

Number of vortons N	CPU time for treecode (s)	CPU time for direct calculation (s)	Maximum relative error	RMS error
500	0.078	0.063	0.00004	0.00000
1000	0.28	0.312	0.00057	0.00011
5000	2.19	16.09	0.00049	0.00012
10,000	5.95	85.47	0.00042	0.00018
50,000	55.7	2753	0.00050	0.00024
100,000	161.3	11,675	0.00042	0.00021
500,000	1064	246,984	0.00052	0.00022
1,000,000	1975	928,859	0.00047	0.00022
2,000,000	4465	3,715,400	0.00048	0.00023
3,000,000	7384	7,436,906	0.00049	0.00023

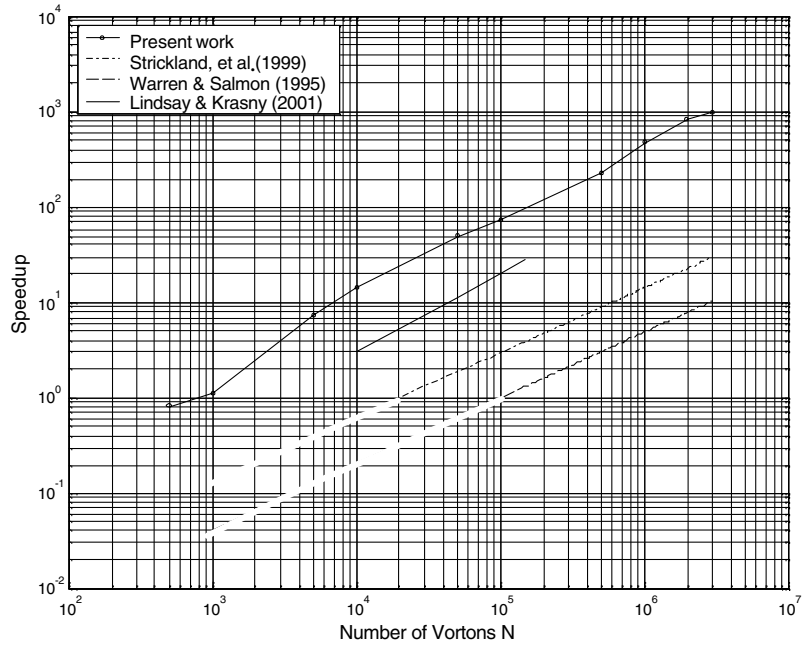

Fig. 4. Comparison of the speedups versus the number of vortons N among the present treecode to those of Strickland et al. [1], Warren and Salmon [2] and Lindsay and Krasny [3] for N vortons and N field points distributed randomly in a cube.

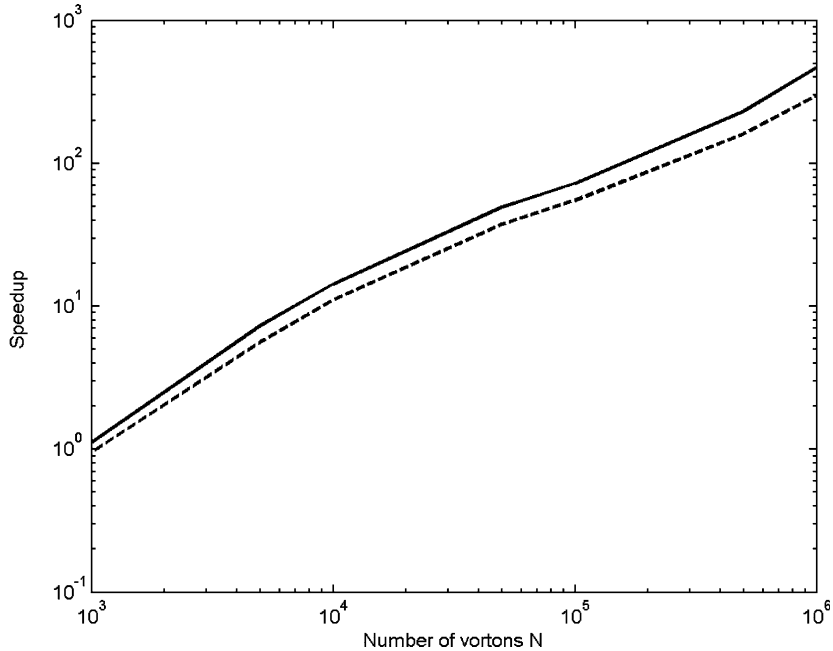

Fig. 5. Comparison of the speedups versus the number of vortons N between the treecodes using real harmonic functions (solid line) and complex harmonic functions (dashed line) for N vortons and N field points distributed randomly in a cube.

randomly in a cube. The tree generation only consumes a small potion of the CPU time, ranging from 1.3% to 0.35% for vortons from $N = 5 \times 10^3$ to 3×10^6, which decreases with the number of vortons. Calculation of the multipole expansion coefficients consumes from 6% to 2% of the CPU time in the range, which decreases with the number of vortons too. The CPU time is consumed predominantly to calculate the target values, and its proportion is from 93% to 98% in the range, and increases with the number of vortons.

Table 3 provides the averaging proportion of the vortons evaluated by the multipole expansion in the first two levels of the source tree for all field leaves for N vortons and N field points distributed randomly in a cube. The averaging proportion of the sources being handled by the multipole expansion is from 34% to 40% at the first level of the tree for $N = 10^4$ to 10^6; and it is from 68% to 73% at the first two levels of the tree for $N = 10^4$ to 10^6. The contribution of two-thirds of the sources has thus been calculated by the multipole expansion at first two levels of the tree averagely.

To examine the performance of the treecode in terms of the expansion order, simulations are performed for 10^5 vortons and 10^5 field points distributed randomly in a cube with various maximum expansion orders $p_{\max}$ from 1 to 20. A uniform expansion order is used as $p_{\max} \leqslant 4$. The variable expansion order is used as $p_{\max} > 4$, and $\theta(p)$ in (13) is set as follows:

$$\theta(p) = 0.75 + 0.2 \cdot (p_{\max} - p) + 0.05 \cdot (p_{\max} - p)^2 \quad \text{for } p = 4, \ldots, p_{\max}. \tag{16}$$

Note here that (14a) and (14b) are two special cases of (16) at $p_{\max} = 6$ and 7, respectively.

Fig. 6(a) shows the treecode CPU time (solid line) versus maximum expansion order $p_{\max}$ comparing the linear dependency of $C = C_1 \cdot p_{\max}$ (dashed line), where C_1 is the CPU time at $p_{\max} = 1$. Comparing the two curves, one can see that the algorithm reduces the CPU time dependency on expansion order p from $O(p^2)$ of the classical treecode to be significantly lower than a linear relation in $p_{\max}$ for $p_{\max} = 2, 3, \ldots, 20$. The maximum relative error and RMS error for velocity versus $p_{\max}$ is shown in Fig. 6(b). The treecode performance for the case is also listed in Table 4. For only one order increase in the CPU time as $p_{\max}$ increases

Table 2
The proportions of the CPU time used for generating the trees, calculating the multipole expansion coefficients, and evaluating the functions for N vortons and N field points distributed randomly in a cube

Number of vortons N	CPU time percentages		
	Tree generation (%)	Multipole expansion coefficients (%)	Function evaluation (%)
5000	1.3	6	93
10000	1	6	93
50,000	0.68	4	95
100,000	0.49	2.9	96
1,000,000	0.47	2.7	97
3,000,000	0.35	2	98

Table 3
The averaging proportion of the vortons handled by the multipole expansion at the first two levels of the tree for N vortons and N field points distributed randomly in a cube

Number of vortons N	Proportion of vortons handled by multipole expansion	
	At level 1	At level 2
10,000	34.0%	34.4%
100,000	38.3%	33.7%
1,000,000	40.3%	32.9%

from 1 to 20, the maximum relative error is reduced by five orders of magnitude (from 2×10^{-2} to 1×10^{-7}), and the RMS error is reduced by six orders of magnitude (from 2×10^{-2} to 3×10^{-8}). With the traditional treecode of $O(p^2)$, the corresponding increase in the CPU time would be two orders of magnitude.

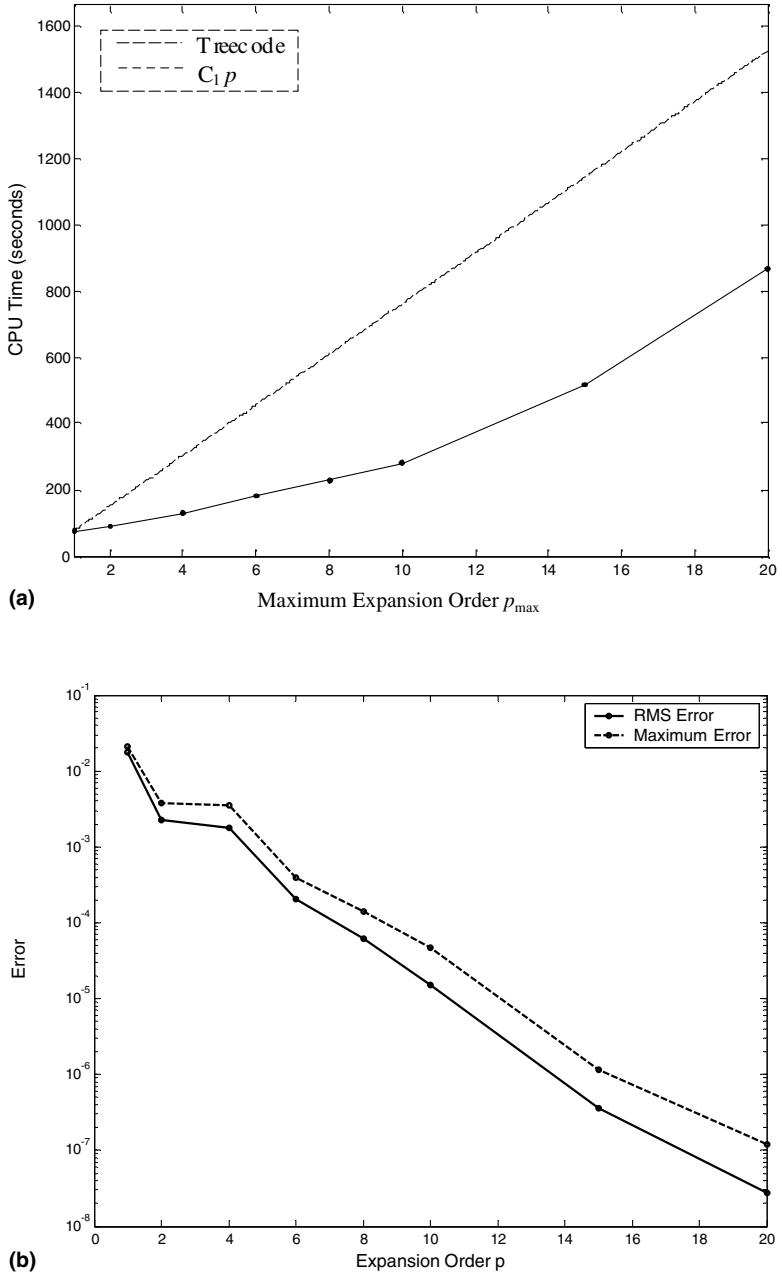

Fig. 6. (a) The CPU time of the treecode and (b) the RMS-error and maximum relative error for velocity versus maximum expansion order p_{max} for 10^5 vortons and 10^5 field points distributed randomly in a cube.

Table 4
Performance of the treecode for 10^5 vortons and 10^5 field points distributed randomly in a cube at various expansion orders. The CPU time for the direct calculation for the case is of 11,200 s

Maximum expansion order p_{max}	CPU time (s)	Maximum relative error	RMS error
1	76.3	0.0209603	0.01771557
2	91.5	0.0037439	0.00226992
4	127	0.0035168	0.00176195
6	181	0.0003878	0.00020270
8	227	0.0001398	0.00006106
10	280	0.0000475	0.00001515
15	519	0.0000011	0.00000035
20	869	0.0000001	0.00000003

5.3. Parallelepiped domains

At last, two cases are considered for N vortons and N field points distributed randomly in a 5:1:1 parallelepiped and a 10:1:1 parallelepiped, respectively. The simulations are performed for various numbers of vortons from $N = 500$ to 2×10^6. The maximum expansion order is chosen at $p_{max} = 7$ and $\theta(p)$ in (13) is set according to (14b). The RMS error and maximum relative error for velocity have been kept at $O(10^{-5})$ for both of the two cases. Fig. 7 displays the speedups of the treecode versus the number of vortons with the revised binary tree (solid line) against that with the oct-tree (dashed line) for the case of a 5:1:1 parallelepiped. As expected, the revised binary treecode reduces the CPU time significantly for this case because of its high adaptiveness to the computational domain. In fact, its speedup is about 2 times higher than that of

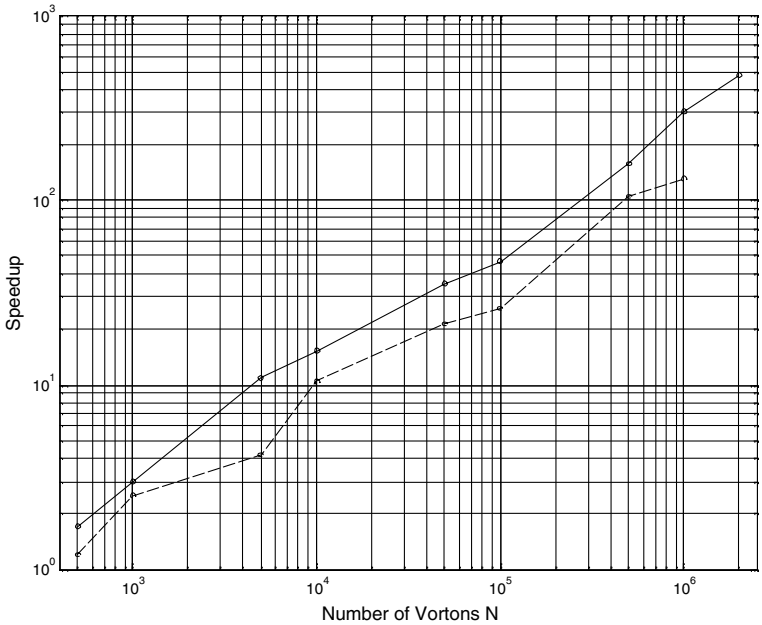

Fig. 7. The speedups of the treecode versus the number of the vortons N with the revised binary tree (solid line) and oct-tree (dash line) for N vortons and N field points distributed randomly in a 5:1:1 parallelepiped.

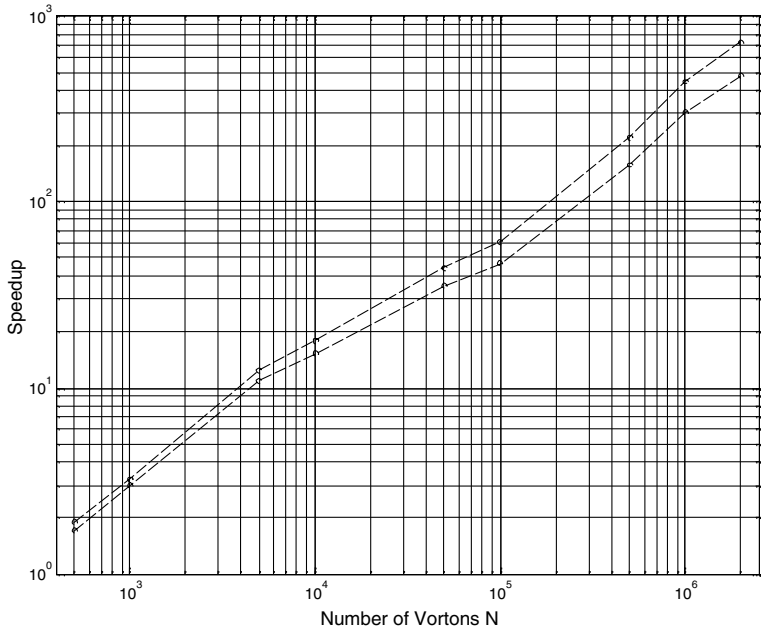

Fig. 8. The speedups of the revised binary treecode versus the number of the vortons N for N vortons and N field points distributed randomly in a 10:1:1 parallelepiped (solid line) and a 5:1:1 parallelepiped (dashed line).

the oct-treecode at $N = 10^6$. The revised binary treecode is faster than the direct method at $N = 500$, of one order of magnitude faster at $N = 5 \times 10^3$, and of two orders of magnitude faster when $N = 3 \times 10^5$. Its speedup is close to 500 times at $N = 2 \times 10^6$.

Fig. 8 compares the speedup of the revised binary treecode for N vortons and N field points distributed randomly in a 10:1:1 parallelepiped (solid line) against that for a 5:1:1 parallelepiped (dashed line). One can see that the speedup for the case of a 10:1:1 parallelepiped is obviously higher, being 1.5 times that of a 5:1:1 parallelepiped at $N = 10^6$. For this case, the revised binary treecode is about 2 times faster than the direct method at $N = 500$, of one order of magnitude faster at $N = 5 \times 10^3$, and of two orders of magnitude faster when $N = 2 \times 10^5$. The speedup is at 700 times at $N = 2 \times 10^6$.

We note here that the vorton distribution of (11) is non-Coulomb interaction, which, in this work, is approximated as a point vortex with a relative error at 1.6×10^{-7} with a cutoff distance at 2.5δ from the vorton centre. As such, significantly more direct calculations have to be performed as compare to pure Coulomb interactions. The CPU time for the multipole expansion has been reduced dramatically by the utilization of the variable expansion order and a new criterion for choosing the order, etc. Because of the above reasons, a large portion of the CPU time is used for the direct-close interactions in the treecode; consequently, the further reducing of the CPU time due to the revised-binary tree is not dramatic. But the speedup of the revised binary treecode as compared to the direct method is significant, which is at 700 times at $N = 2 \times 10^6$ and RMS Error $= O(10^{-5})$ for the case of a 10:1:1 parallelepiped. By the way, the revised binary tree devised here is mainly for the parallel efficiency and parallelization easiness of a treecode.

6. Summary and conclusions

Three important improvements are described to the treecode, which are illustrated and applied to calculate the velocity induced by vortex elements. Firstly, the multipole expansion is implemented based on the

real spherical harmonic functions rather than the complex ones deployed in the field, which reduces about one-third of the CPU time.

Secondly, the expansion order is given in terms of the ratio of the distance of a field point to a source box to the box size, which reflects the relative error of the multipole expansion. The variable order treecode has the following features:

(1) The contribution of a large portion of the sources has been evaluated by the multipole expansion at low levels of the source tree. The averaging proportion of the sources handled by the multipole expansion is from 34% to 40% at the first level of the tree for $N = 10^4$ to 10^6; and it is from 68% to 73% at the first two levels of the tree. The contribution of two-thirds of the sources has been calculated by the multipole expansion at first two levels of the tree averagely.
(2) With a higher order expansion deployed for source boxes near a field leaf, the minimum well-separation distance for deploying the expansion is reduced and the direct evaluation is limited to smaller number of sources.
(3) It decreases the CPU time dependency on expansion order p. The simulations have indicated that the dependency drops from $O(p^2)$ of the classical treecode to be significantly lower than a linear dependency in $p_{\max}$ for $p_{\max} = 2, 3, \ldots, 20$, where $p_{\max}$ is the maximum expansion order used in the variable order expansion.
(4) It maintains the high parallel efficiency of the classical treecode.

The efficiency of the algorithm is mainly due to the utilization of the variable order expansion and the new criterion for choosing the expansion order.

Thirdly, a revised binary tree is built by performing the bisections three times in a row at each tree level, discarding the boxes generated in the first two bisections and remaining only the boxes generated in the last one. The revised binary tree has the following features:

(1) The revised binary tree avoids the disadvantages of a binary tree having about 1.75 times more boxes, three time more levels and consequently requiring more CPU time.
(2) As compared to an oct-tree, this tree has much higher adaptiveness to the computational domain and source distribution.
(3) In this tree, the boxes at a given level contain essentially the same number of sources (within ± 1), thus has perfect load balancing for performing the parallelization.
(4) It is an adaptive tree but it is a complete tree whose leaves all have the same depth, the parallelization of a revised binary treecode is thus as simple as that of a non-adaptive approach.

Simulations have been performed for N vortons and N field points distributed randomly in a cube, a 5:1:1 parallelepiped, and a 10:1:1 parallelepiped, using the oct-treecode and revised binary treecode, respectively. The algorithm is an order faster than those of Strickland et al. [1], Warren and Salmon [2], and Lindsay and Krasny [3]. Simulations also demonstrate the high efficiency of the revised binary treecode for an inhomogeneous source distribution.

Finally, the variable order revised binary treecode developed here can be applied for the gravitational and electrostatic potentials straightforwardly; it can also be implemented for the screened electrostatic potential and general power-law interactions in molecular dynamics. The revised binary tree and the multipole expansion based on the real harmonic functions devised here may be applied to the fast multipole expansion too.

Acknowledgements

This work was mainly carried out while the author was at the DSO National Laboratories. He wants to express his thanks to Mr. E.K. Png, Dr. Y.W. Chan, Dr. J.K. Tan, and Mr. N.H. Teng at DSO, and Dr. Y. Cao and Dr. B. Maskew at Analytical Methods, Inc., for their supports and suggestions.

The author also wants to express his sincere thanks to the referees of this paper for their valuable suggestions.

References

[1] J.H. Strickland, L.A. Gritzo, R.S. Baty, G.F. Homicz, Fast multipole solvers for 3D radiation and fluid flow problems, ESAIM: Proceedings 7 (1999) 408.
[2] M.S. Warren, J.K. Salmon, A portable parallel program, Comput. Phys. Commun. 87 (1995) 266.
[3] K. Lindsay, R. Krasny, A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow, J. Comput. Phys. 172 (2001) 879.
[4] J.E. Barnes, P. Hut, A hierarchical $O(N \log N)$ force-calculation algorithm, Nature 324 (4) (1986) 446.
[5] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys. 73 (1987) 325.
[6] L. Greengard, Fast algorithms for classical physics, Science 265 (1994) 909.
[7] R. Beatsonm, L. Greengard, A short course on fast multipole methods in wavelets, in: Ainsworth et al. (Eds.), Multilevel Methods & Elliptic PDFs, Oxford University Press, Oxford, 1997, pp. 1–37.
[8] H. Cheng, L. Greengard, V. Rokhlin, A fast adaptive multipole algorithm in three dimensions, J. Comput. Phys. 155 (2) (1999) 468.
[9] G. Blelloch, G. Narlikar, A practical comparison of N-body algorithms, Parallel Algorithms, Series in Discrete Mathematics and Theoretical Computer Science, vol. 30, 1997.
[10] R. Capuzzo-Dolcetta, P. Miocchi, A comparison between the fast multipole algorithm and the tree-code to evaluate gravitational forces in 3-D, J. Comput. Phys. 143 (1) (1998) 29.
[11] Z.H. Duan, R. Krasny, An Ewald summation based multipole method, J. Comput. Phys. 113 (9) (2000) 3492.
[12] Z.H. Duan, R. Krasny, An adaptive treecode for computing nonbonded potential energy in classical molecular systems, J. Comput. Chem. 22 (2) (2001) 184.
[13] H. Petersen, D. Soelvason, J. Perram, E. Smith, The very fast multipole method, J. Chem. Phys. 101 (10) (1994) 8870.
[14] T. Schlick, Molecular Modeling and Simulation: An Interdisciplinary Guide, Springer, Berlin, 2002.
[15] T. Schlick, R.D. Skeel, A.T. Brunger, L.V. Kale, J.A. Board, J. Hermans, K. Schulten, Algorithmic challenges in computational molecular biophysics, J. Comput. Phys. 151 (1) (1999) 9.
[16] R. Zhou, B.J. Berne, A new molecular dynamics method combining the reference system propagator algorithm with a fast multipole method for simulating proteins and other complex systems, J. Chem. Phys. 103 (21) (1995) 9444.
[17] J.K. Salmon, M.S. Warren, Skeletons from the treecode closet, J. Comput. Phys. 111 (1) (1994) 136.
[18] J.K. Salmon, M.S. Warren, Parallel out-of-core methods for n-body simulation, in: 8th SIAM Conference on Parallel Processing for Scientific Computing, 1997.
[19] U. Becciani, V. Antonuccio-Delogu, M. Gambera, A modified parallel tree code for N-body simulation of the large-scale structure of the universe, J. Comput. Phys. 163 (1) (2000) 118.
[20] W. Dehnen, A hierarchical $O(N)$ force calculation algorithm, J. Comput. Phys. 179 (1) (2002) 27.
[21] K. Nabors, F.T. Korsmeyer, F.T. Leighton, J. White, Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory, SIAM J. Sci. Comput. 15 (3) (1994) 714.
[22] A. Grama, V. Kumar, A. Sameh, Parallel hierarchical solvers and preconditioners for boundary element methods, SIAM J. Sci. Comput. 20 (1) (1998) 337.
[23] A. Grama, V. Sarin, A. Sameh, Improving error bounds for multipole-based treecodes, SIAM J. Sci. Comput. 21 (5) (2000) 1790.
[24] C.R. Anderson, C. Greengard, The vortex ring merger problem at infinite Reynolds number, Commun. Pure Appl. Math. 42 (1989) 1123.
[25] J.H. Strickland, R.S. Baty, An overview of fast multipole methods, SAND95-2405, 1995.
[26] J.H. Strickland, R.S. Baty, A pragmatic overview of fast multipole methods, Lect. Appl. Math. 32 (1996) 807.
[27] G.S. Winckelmans, J.K. Salmon, M.S. Warren, A. Leonard, Application of fast parallel and sequential tree codes to computing three-dimensional flows with the vortex element and boundary element methods, ESAIM: Proceedings 1 (1996) 225.
[28] L.A. Gritzo, J.H. Strickland, A gridless solution of the radiative transfer equation for fire and combustion calculations, Combust. Theor. Model 3 (1) (1999) 159.
[29] J.P. Collins, A.A. Dimas, P.S. Bernard, A parallel adaptive fast multipole method for high performance vortex method based simulations, in: Proceedings of the ASME Fluids Engineering Division, FED-vol. 250, 1999.
[30] J.S. Marshall, J.R. Grant, A.A. Gossler, S.A. Huyer, Vorticity transport on a Lagrangian tetrahedral mesh, J. Comput. Phys. 161 (2000) 85.
[31] J.H. Walther, P. Koumoutsakos, Three-dimensional vortex methods for particle-laden flows with two-way coupling, J. Comput. Phys. 167 (2001) 39.

[32] P. Ploumhans, G.S. Winckelmans, J.K. Salmon, A. Leonard, M.S. Warren, Vortex methods for direct numerical simulation of three-dimensional bluff body flows: application to the sphere at $Re = 300$, 500, and 1000, J. Comput. Phys. 178 (2) (2002) 427.
[33] J.D. Eldredge, T. Colonius, A. Leonard, A vortex particle method for two-dimensional compressible flow, J. Comput. Phys. 179 (2) (2002) 371.
[34] N.R. Clarke, O.R. Tutty, Construction and validation of a discrete vortex method for the two-dimensional incompressible Navier–Stokes equations, Comput. Fluids 23 (6) (1994) 751.