Available online at www.sciencedirect.com
ScienceDirect
Journal of Computational Physics 226 (2007) 17–28
JOURNAL OF COMPUTATIONAL PHYSICS
www.elsevier.com/locate/jcp

Adaptive spatial decomposition in fast multipole method

J.M. Zhang [a,*], Masa. Tanaka [b]

[a] State Key Laboratory of Advanced Design and Manufacturing for Vehicle Body, College of Mechanical and Automotive Engineering, Hunan university, Changsha 410082, China
[b] Faculty of Engineering, Shinshu University, Nagano, Japan

Received 13 September 2006; received in revised form 22 February 2007; accepted 25 March 2007
Available online 18 April 2007

Abstract

This work presents a new adaptive node-cluster algorithm for fast multipole method. In the algorithm, we use rectangular boxes instead of cubes, subdivide a box based on its shape, and tighten the child boxes at each subdivision step. More importantly, we determine the number of expansion terms in multipole to local translations according to the distance between the two interaction boxes. Our method is tested using benchmark examples for three-dimensional potential problems. The results obtained show that the new algorithm can solve a problem with 100 thousands nodes in about 20 min, and runs nearly three times faster than the standard algorithm. The proposed algorithm is especially suitable for treating slender and shell-like structures.

Keywords: Adaptive node-cluster; Tree data structure; Fast multipole method; Hybrid boundary node method

1. Introduction

The fast multipole method (FMM) [1] is regarded as one of the top 10 algorithms of the 20th century. It is capable of achieving fast multiplication of particular dense matrices with vectors, and it allows for the reduction of memory complexity. Generally, the FMM reduces the computational cost for the matrix–vector multiplication from $O(N^2)$ to $O(N)$, where N is the total number of unknowns, thus making possible scientific and engineering computations of large scale problems.

The FMM uses multipole expansions (in term of series) to approximate the effects of a distant group of particles on a local group, and translations between these expansions. Another aspect of the FMM is that it uses a hierarchical decomposition of space to define ever-larger groups as distances increase. In 3D cases, an oct-tree decomposition is usually employed. The multipole expansions and translations are orchestrated within the tree in an effective way to obtain an algorithm with $O(N)$ asymptotic complexity.

The major obstacle in achieving reasonable efficiency with high accuracy is the large number of the multipole to local translations (M2L). To overcome this obstacle, Greengard and Rokhlin [2] proposed a

* Corresponding author. Tel.: +81 26 269 5123; fax: +81 26 269 5124.
E-mail address: zhangjianm@gmail.com (J.M. Zhang).

0021-9991/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2007.03.032

new diagonal form, which reduces the M2L cost from $O(p^4)$ to $O(p^2)$, where p is the number of terms in the truncated expansion series. Another way for reducing the cost of translation operators is to lower the number of M2L operations by using a new tree data structure. Anderson [3] studied systematically how a spatial data structure influences the performance of FMM, and concluded that a binary, spatially balanced decomposition tree with tight bounds is the best tree data structure for FMM. Recently, Urago et al. [4] implemented this idea in a FMM simulation of electrostatic field, and gained an efficiency improvement of two to three times faster than the standard algorithm. To better balance the near-field and far-field work, White and Head-Gordon [5] proposed a concept of fractional tiers (depth of tree), which was implemented by using a changeable number of particles in the leaves of the tree. Their method gives a speedup approaching two times.

The FMM was first introduced as a fast solution method in astrophysics for simulation of N-body systems in which the interactions between the bodies are gravitational. Because of the computational analogy between the force evaluation for the N-body problem and the matrix–vector multiplication, the FMM is widely employed in conjunction with iterative solvers to accelerate the solutions of elliptic partial differential equations (PDEs) through the boundary integral equation (BIE). We are working on the latter case. We have proposed a boundary type meshless method, called hybrid boundary node method (HdBNM) [6,7], and combined the method with FMM [8]. Like the boundary element method (BEM), the HdBNM only needs points on the surface on a computational domain. The boundary-only distribution of points gives chances to find a special tree data structure which is most efficient in grouping the points into well-separated divisions.

In this paper, we proposed a new adaptive node-cluster algorithm for the fast HdBNM. In our algorithm, we use rectangular boxes instead of cubes, and subdivide a box according to its shape. Therefore, the tree data structure in the new algorithm is neither an oct-tree nor a binary tree. The number of offspring of a parent box is dependent on the shape of the box. We also tighten the child boxes at each subdivision step, and generalize the downward pass algorithm so that M2L translations can be performed among the child boxes of a single parent box (brother boxes). More importantly, we determine the value of p for the M2L translations by the distance between the two interaction boxes. Numerical examples presented show that the new algorithm can solve a problem with 100 thousands nodes using about 20 min, and it runs nearly three times faster than the standard algorithm for solving equation

2. Review of the hybrid boundary node method

In this paper, we combine the FMM with the hybrid boundary node method (HdBNM) to demonstrate the proposed algorithm. In this section, therefore, we give a brief description of the HdBNM. The HdBNM is based on the modified variational principle. With the 3-D steady state heat conduction problem as an example, the independent functions in the modified functional are the temperature ϕ in the domain, the boundary temperature $\tilde{\phi}$ and boundary normal flux $\tilde{q}$. Consider a domain Ω enclosed by $\Gamma = \Gamma_\phi + \Gamma_q$ with prescribed potential $\bar{\phi}$ and normal flux $\bar{q}$ on the boundary portions Γ_ϕ and Γ_q, respectively. The corresponding variational functional Π_{AB} is defined as

$$\Pi_{AB} = \int_\Omega \frac{1}{2}\phi_{,i}\phi_{,i}\,\mathrm{d}\Omega - \int_\Gamma \tilde{q}(\phi - \tilde{\phi})\,\mathrm{d}\Gamma - \int_{\Gamma_q} \bar{q}\tilde{\phi}\,\mathrm{d}\Gamma, \tag{1}$$

where the boundary temperature $\tilde{\phi}$ satisfies the essential boundary condition, i.e., $\tilde{\phi} = \bar{\phi}$ on Γ_ϕ .

Suppose that N nodes are well distributed on the bounding surface of the domain. The temperature inside the domain is then approximated using fundamental solutions as follows:

$$\phi = \sum_{I=1}^{N} \phi_I^s x_I. \tag{2}$$

At a boundary point, it follows that the normal flux is given by

$$q = \sum_{I=1}^{N} \frac{\partial \phi_I^s}{\partial n} x_I, \tag{3}$$

where ϕ_I^s is the fundamental solution with the source at a node $\mathbf{s}_I$; and x_I are unknown parameters. For 3-D potential problems, the fundamental solution can be written as

$$\phi_I^s = \frac{1}{4\pi r(Q,\mathbf{s}_I)}, \tag{4}$$

where Q is a field point, and $r(Q,\mathbf{s}_I)$ is the distance between Q and $\mathbf{s}_I$.

The boundary temperature $\tilde{u}$ and the normal flux $\tilde{q}$ are interpolated by the moving least-square approximation (MLS) [7] as follows:

$$\tilde{\phi}(\mathbf{s}) = \sum_{I=1}^{N} \Phi_I(\mathbf{s})\hat{\phi}_I, \tag{5}$$

$$\tilde{q}(\mathbf{s}) = \sum_{I=1}^{N} \Phi_I(\mathbf{s})\hat{q}_I. \tag{6}$$

In the above equations, $\Phi_I(\mathbf{s})$ is the shape function of MLS; $\hat{\phi}_I$ and $\hat{q}_I$ are nodal values of temperature and normal flux, respectively.

With the local sub-domain around each node taken into consideration, the stationary conditions can be obtained by taking variations in Eq. (1) with respect to the independent variables. This gives the following set of equations:

$$\mathbf{U}\mathbf{x} = \mathbf{H}\hat{\phi}, \tag{7}$$

$$\mathbf{Q}\mathbf{x} = \mathbf{H}\hat{\mathbf{q}}, \tag{8}$$

where $\mathbf{U}$, $\mathbf{Q}$ and $\mathbf{H}$ are defined as:

$$U_{IJ} = \int_{\Gamma_I} \phi_J^s(Q,\mathbf{s}_J)v_I(Q)\,\mathrm{d}\Gamma, \tag{9}$$

$$Q_{IJ} = \int_{\Gamma_I} \frac{\partial\phi_J^s(Q,\mathbf{s}_J)}{\partial n(Q)}v_I(Q)\,\mathrm{d}\Gamma, \tag{10}$$

$$H_{IJ} = \int_{\Gamma_I} \Phi_J(Q)v_I(Q)\,\mathrm{d}\Gamma, \tag{11}$$

where v_I is a weight function, Γ_I is a regularly shaped local region around a given node $\boldsymbol{s}_I$ in the parametric representation space of the boundary surface. Therefore, the integrals in Eqs. (9)–(11) can be calculated without using boundary elements (for details refer to [7]).

For a well-posed problem, either $\hat{\phi}_I$ or $\hat{q}_I$ is known at a node $\boldsymbol{s}_\mathrm{I}$ on the boundary. Thus, Eqs. (7) and (8) can be solved for the unknown parameters $\mathbf{x}$. Then, by back-substitution of $\mathbf{x}$ into Eqs. (7) and (8), the boundary unknowns are obtained for both temperature and normal flux by solving Eqs. (7) and (8) with $\mathbf{H}$ being the coefficient matrix.

The coefficient matrices $\mathbf{U}$ and $\mathbf{Q}$ are dense and unsymmetrical. It requires $O(N^2)$ memory to store them and $O(N^3)$ CPU time to solve them if a direct solver is employed. When we use an iterative solver, such as GMRES, the most time-consuming part of computation will be the matrix–vector multiplication in each iteration step. Considering an iteration vector $\mathbf{x}^k$ at the kth step, the matrix–vector multiplication at the $k+1$th step is

$$x_I'^{k+1} = \sum_{J=1}^{N} \int_{\Gamma_I} \phi_J^s v_I(Q)x_J^k\,\mathrm{d}\Gamma \tag{12}$$

or

$$x_I'^{k+1} = \sum_{J=1}^{N} \int_{\Gamma_I} \frac{\partial\phi_J^s}{\partial n}v_I(Q)x_J^k\,\mathrm{d}\Gamma, \tag{13}$$

where $\mathbf{x}'^{k+1}$ is a temporary vector from which $\mathbf{x}^{k+1}$ is then computed according to the iteration scheme of the solver. Direct computation of Eqs. (12) and (13) gives an $O(N^2)$ algorithm. The FMM can be employed to reduce the complexity to $O(N)$.

3. Review of the fast multipole method

The FMM mainly uses three addition theorems which are briefly explained below.

First addition theorem: Define solid spherical harmonics $R_n^m(\mathbf{r})$ and $S_n^m(\mathbf{r})$ as [9]

$$R_n^m(\mathbf{r}) = \frac{1}{(n+m)!}P_n^m(\cos\alpha)e^{im\beta}r^n,$$

$$S_n^m(\mathbf{r}) = (n-m)!P_n^m(\cos\alpha)e^{im\beta}\frac{1}{r^{n+1}}.$$

Here (r,α,β) is spherical coordinates of the point $\mathbf{r}$; $P_n^m(\cos\alpha)$ is the associated Lengendre function of integer order m and degree n. Let $\mathbf{r}_1$ and $\mathbf{r}_2$ be two points with spherical coordinates (r_1,α_1,β_1) and (r_2,α_2,β_2), respectively. It follows that:

$$\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|} = \begin{cases} \sum_{n=0}^{\infty}\sum_{m=-n}^{n} R_n^m(\mathbf{r}_1)\overline{S_n^m(\mathbf{r}_2)}, & |\mathbf{r}_1| < |\mathbf{r}_2|, \\ \sum_{n=0}^{\infty}\sum_{m=-n}^{n} R_n^m(\mathbf{r}_2)\overline{S_n^m(\mathbf{r}_1)}, & |\mathbf{r}_1| > |\mathbf{r}_2|. \end{cases} \tag{14}$$

In the above equation, the overhead bar means the complex conjugate of a complex number.

Second addition theorem: If $\mathbf{r}_1$ and $\mathbf{r}_2$ are two vectors such that $|r_1| > |r_2|$, then

$$S_n^m(\mathbf{r}_1-\mathbf{r}_2) = \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'} \overline{R_{n'}^{m'}(\mathbf{r}_2)}S_{n+n'}^{m+m'}(\mathbf{r}_1). \tag{15}$$

Third addition theorem: If $\mathbf{r}_1$ and $\mathbf{r}_2$ are two arbitrary vectors, then

$$R_n^m(\mathbf{r}_1-\mathbf{r}_2) = \sum_{n'=0}^{n}\sum_{m'=-n'}^{n'} R_{n'}^{m'}(-\mathbf{r}_2)R_{n-n'}^{m-m'}(\mathbf{r}_1). \tag{16}$$

Instead of treating interactions with each of the distant nodes individually, the FMM computes cell–cell interactions. Consider two cells C_a and C_b, which contain N_a and N_b nodes, respectively. The computational complexity of a standard algorithm for the mutual interactions between the two groups is of order $O(N_a \times N_b)$ (Fig. 1a). In the cell–cell strategy, however, it is reduced to $O(N_a + N_b)$ (Fig. 1b).

Substituting Eq. (4) into Eq. (12) and using the first addition theorem, with the summation over the nodes included in C_b, we obtain

$$\sum_{J=1}^{N_b}\int_{\Gamma_I}\phi_J^s v_I(Q)x_J^k\,\mathrm{d}\Gamma = \sum_{n=0}^{\infty}\sum_{m=-n}^{n}\int_{\Gamma_I}\frac{1}{4\pi}\overline{S_n^m}(\vec{O_2Q})\,v_I(Q)\,\mathrm{d}\Gamma M_n^m(O_2), \tag{17}$$

where the *coefficients of multipole expansion* $M_n^m(O_2)$ are defined by

$$M_n^m(O_2) = \sum_{J=1}^{N_b} R_n^m(\vec{O_2\mathbf{s}_j})\,x_J^k. \tag{18}$$

In Eqs. (17) and (18), the points O_1, O_2, Q and $\mathbf{s}_J$ are shown in Fig. 1c. Using further the second addition theorem, Eq. (17) becomes

$$\sum_{J=1}^{N_b}\int_{\Gamma_I}\phi_J^s v_I(Q)x_J^k\,\mathrm{d}\Gamma = \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'}\int_{\Gamma_I}\frac{1}{4\pi}R_{n'}^{m'}(\vec{O_1Q})\,v_I(Q)\,\mathrm{d}\Gamma L_{n'}^{m'}(O_1), \tag{19}$$

where the *coefficients of local expansion* $L_{n'}^{m'}(O_1)$ are given by

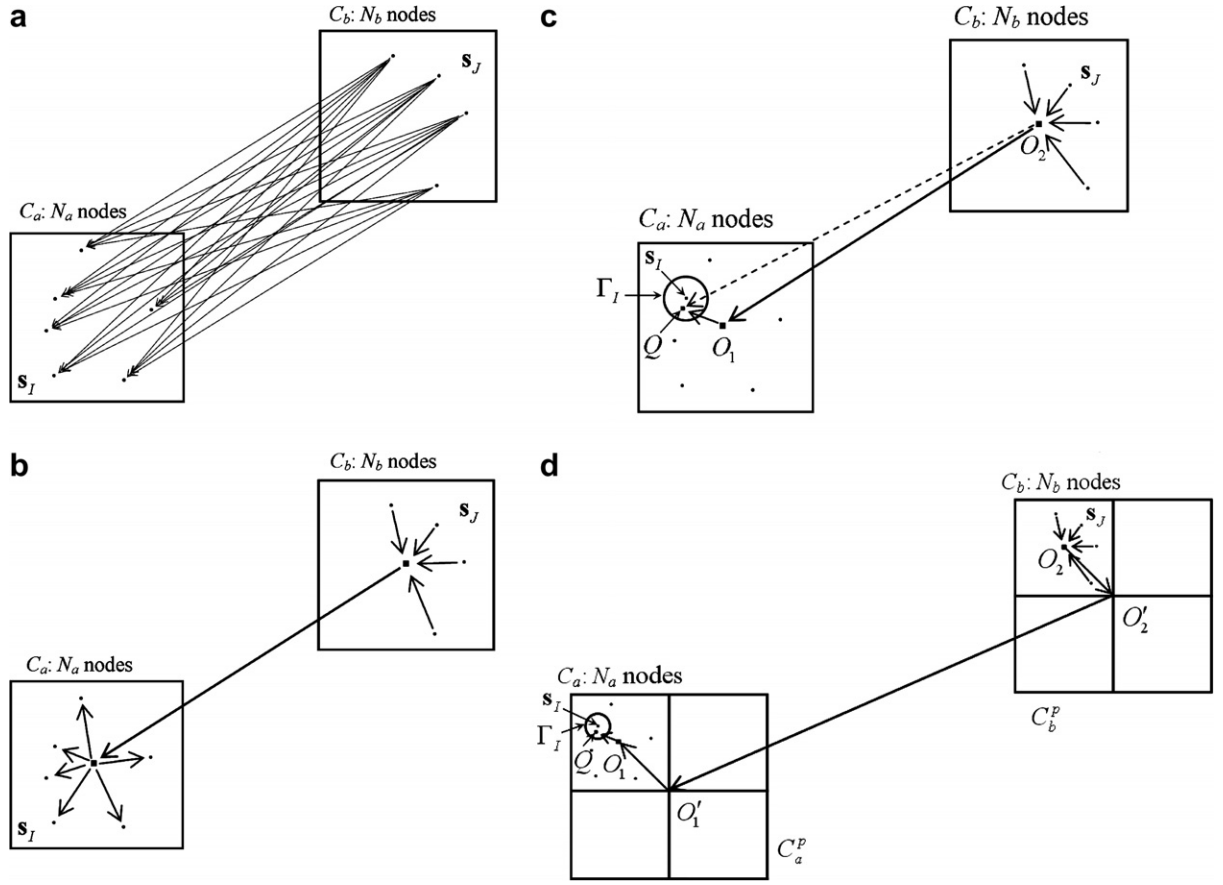

Fig. 1. Interaction between two cells.

$$L_{n'}^{m'}(O_1) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (-1)^{n'} \overline{S_{n+n'}^{m+m'}}(\vec{O_1Q})\, M_n^m(Q_2). \tag{20}$$

Eq. (20) is known as the *multipole to local* (M2L) translation, as it transforms the *coefficients of multipole expansion* of C_b to the *coefficients of local expansion* of C_a.

Suppose that C_a and C_b are obtained by subdividing other two larger cells C_a^p and C_b^p, known as the parent cells of C_a and C_b, respectively. Assume that C_a^p and C_b^p are still far away from each other (see Fig. 1d). We can then transform the *coefficients of multipole expansion* of C_b to that of C_b^p (M2M) using the third addition theorem, transform the *coefficients of multipole expansion* of C_b^p to *local moments* of C_a^p (M2L), and finally to *coefficients of local expansion* of C_a (L2L) using the third addition theorem again. Therefore, Eq. (20) becomes

$$L_{n'}^{m'}(O_1) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} R_{n-n'}^{m-m'}(\vec{O_1Q})\, L_n^m(Q_1') \tag{21}$$

and

$$L_n^m(O_1') = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (-1)^n \overline{S_{n+n'}^{m+m'}}(\vec{O_1'O_2'})\, M_{n'}^{m'}(Q_2'), \tag{22}$$

$$M_{n'}^{m'}(Q_2') = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} R_n^m(\vec{O_2'O_2})\, M_{n-n'}^{m-m'}(Q_2). \tag{23}$$

The above process can be recursively repeated until it reaches the root cell which contains the entire computational domain. In the above process, the addition theorems are used to separate the source and target points in the fundamental solution and in the solid spherical harmonics, so that the *coefficients of multipole expansion* and *local expansion* are related only to each individual cells. Therefore, these coefficients can be calculated independently and can be aggregated into ones to represent temperature due to ever larger groups of nodes. Moreover, once calculated, they can be reused for other cell–cell interactions.

4. Tree construction

In the previous section, we have described the process of cell–cell interaction. We have seen that the two points in two-point functions can be separated freely by addition theorems. All the resulted coefficients of expansion can be calculated independently. This allows for the freedom to arrange these computations in order to achieve better efficiency. In the FMM, actually, the cell–cell interaction is not performed separately for each pair of well-separated cells. An elaborate algorithm has been designed. This algorithm is facilitated by a tree data structure, which hierarchically decomposes the entire region into cells at different levels. There is a tremendous flexibility in the choice of tree data structure that could be used in the algorithm. To enhance the accuracy of the computation, it is important to note that the cells have roughly the same size in all directions. It is also desirable that the cells chosen reflect the geometry of the computational domain as accurately as possible.

The standard FMM algorithm uses an oct-tree. The entire computational domain is assumed to lie inside a cube, which is referred as the root cube at level 0. The oct-tree is constructed by recursively subdividing the cubes into eight sub-cubes by splitting each cube at the geometrically central point. The cubes at level $l+1$ are obtained from cubes at level l, where the eight sub-cubes at level $l+1$ are considered children of the cube at level l. The subdivision continues until cubes contain less than a given number of particles (e.g. boundary nodes in HdBNM). If a child cube does not contain any node (that is, it is empty), it is deleted. A childless cube is called a leaf.

With the tree, the FMM consists of two basic steps: *upward pass* and *downwards pass*. During the upward pass, the *coefficients of multipole expansion* are summed from its children using the M2M translation for each non-leaf cube. In the downwards pass, the tree is traversed from the root to leaves to compute the *coefficients of local expansion*. For each cube C, these coefficients are the sums of two parts. Firstly, the L2L translation collects the coefficients of C's parent. Secondly, the M2L translation collects the *coefficients of multipole expansion* of the cubes which are the children of the neighbors of C's parent but are not adjacent to C (these cubes compose the interaction list of C). Finally, for each leaf, the far interaction, which is evaluated using the *coefficients of local expansion* at this cube is combined with the near interaction evaluated by iterating over all the source nodes in the neighborhood of the leaf cube to obtain the entire sum in Eq. (12).

5. An adaptive tree

Usually, there are a large number of M2L translations for each cube at every level. Therefore, the computational cost of FMM is dominated by the M2L translation. One way to reduce the number of M2L translations is to find a better tree data structure [3–5]. The hypothesis is that a tree data structure that matches the geometry of the computational domain and has shallow depth will allow the computation to be done with a smaller number of M2L translations. It is obvious that the oct-tree does not necessarily match the structures commonly used in engineering (a shell-like structure or a slender object, for example) since the oct-tree is constructed by choosing splitting planes oblivious to node distribution. As the cost of tree construction is minor, it is possible to use a tree data structure that is more expensive to construct than the oct-tree and still achieve a net gain in performance.

Anderson [3] has shown that a binary tree has advantages over the oct-tree in matching arbitrary structures. In the binary tree, a non-leaf box has two child boxes, and subdivision of a box is always in the direction of the longest side. Urago et al. [4] implemented the binary tree, and demonstrated that the number of M2L translations was reduced by three folds.

In this paper, we attempt to establish an adaptive tree that can match structures of arbitrary geometries. However, our aim is not to reduce the number of M2L translation, but to obtain smaller values of p in M2L translations under the same precision by decomposing the computational domain into more compact and well-separated cells. The adaptive tree is based on the standard oct-tree, but differs in the following aspects:

1. Instead of using cubes, the adaptive tree uses rectangular boxes to decompose the computational domain. It is believed that a rectangular box is more flexible in matching structures.
2. In contrast to the oct-tree, where subdivision of a cube is oblivious to geometry, a box in the adaptive tree is split into child boxes based on its shape. Without loss of generality, let L_1, L_2, and L_3 be the side lengths of a box in the three coordinate directions, respectively, and suppose $L_1 > L_2 > L_3$. We subdivide the box according to the ratios between the values of the side lengths. More precisely, we consider three cases below:
   (1) When $L_1/L_2 > 1.5$, we split the box in the direction of the longest side (see Fig. 2a). Moreover, the number of child boxes, n, also depends on the ratio, L_1/L_2. If $L_1/L_2 > 7.5$, $n = 8$; else $n = \mathrm{Int}(L_1/L_2)+1$, where Int( ) refers to the integer part of a real number.
   (2) When $L_1/L_2 < 1.5$ and $L_2/L_3 > 1.5$, we split the box in the two directions shown in Fig. 2b. The number of child boxes is fixed to 4.
   (3) When $L_1/L_2 < 1.5$ and $L_2/L_3 < 1.5$, we split the box in the three directions shown in Fig. 2c. The number of child boxes is fixed to 8.
3. Boxes of the adaptive tree are tightened at each subdivision step. Boxes with tightened bounds have been investigated by Anderson [3], and are proven to be more efficient in separating a node set into well-separated clusters. The oct-tree chooses to subdivide the cubes when the cells are split. This method is referred to as *loose bounds*. In adaptive tree, on the other hand, we always use a smallest box to enclose the cluster of boundary nodes. This is referred to as *tight bounds*. When cells are split, the location (and even the dimension) of the split depends upon the bounding faces of the box. This means that the tight bound and loose bound tree for the same set of nodes will have a different structure. Note that when tight bounds are used, the bounding box is computed before the cell is split. This is different from using loose bounds for computing all of the splits and then tightening the boxes. As mentioned in [3], tight bounds may hurt the performance of the algorithm for dense particle based systems. In our case, however, as the nodes are only distributed on the surface of a body, using tight bounds will increase the distance between two interacting cells, thus make them more well-separated.

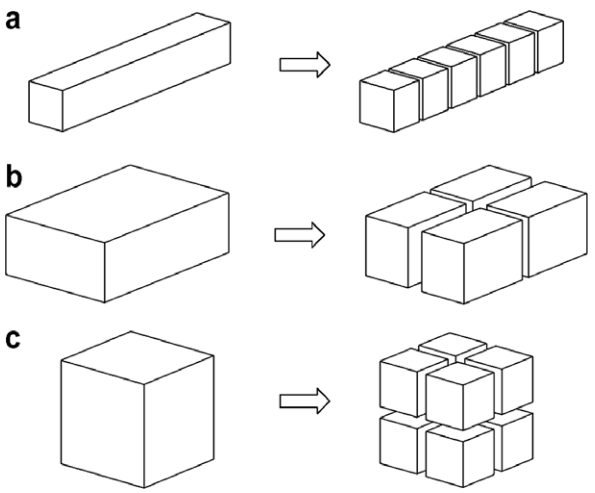

Fig. 2. Subdivision of a box.

4. A more generalized *downward pass* algorithm is designed to allow M2L among the child boxes of a single parent box. The standard algorithm always treats the child boxes of a parent box as neighbors. This is no longer valid for the adaptive tree (see Fig. 2a).
5. In practical computation, the infinite series in Eqs. (19)–(23) are truncated after p terms. The standard algorithm uses a fixed p for all series. Some researchers have also investigated variable orders of multipole expansions [10]. In this paper, we use adaptive values of p for the series in all M2L translations. The value is determined by the distance between each pair of interacting cells and their sizes. The error estimation of the truncated series has been estimated in the original FMM paper [2] as well as in other papers, including [11,12].

According to Eq. (5.7) of Ref. [2], we can write the following equivalent equation for the case of HdBNM:

$$\left|\sum_{J=1}^{N_b}\phi_J^s x_J^k - \sum_{n'=0}^{p}\sum_{m'=-n'}^{n'}\frac{1}{4\pi}R_{n'}^{m'}(\vec{O_1Q})\,L_{n'}^{m'}(O_1)\right| \leqslant \left(\frac{\sum_{J=1}^{N_b}x_J^k}{\rho-2a}s\right)\left(\frac{a}{\rho-a}\right)^{p+1}, \tag{24}$$

where a is the maximum radius of the two spheres that enclose the two interacting cells, respectively, and ρ is the distance between the centers of the two boxes. Weighting with v_I and integrating on Γ_I (see Eq. (9)), we have

$$\left|\sum_{J=1}^{N_b}\int_{\Gamma_I}\phi_J^s v_I(Q)x_J^k\,\mathrm{d}\Gamma - \sum_{n'=0}^{p}\sum_{m'=-n'}^{n'}\int_{\Gamma_I}\frac{1}{4\pi}R_{n'}^{m'}(\vec{O_1Q})\,v_I(Q)\,\mathrm{d}\Gamma L_{n'}^{m'}(O_1)\right| \leqslant \left(\frac{\sum_{J=1}^{N_b}\int_{\Gamma_I}v_I(Q)x_J^k\,\mathrm{d}\Gamma}{\rho-2a}\right)\left(\frac{a}{\rho-a}\right)^{p+1}. \tag{25}$$

Therefore, the error introduced by the multipole to local translation is mainly bounded by the term $(a/(\rho-a))^{p+1}$. In order to get the same accuracy of the standard FMM, suppose that, in the standard FMM, the fixed value of p is p_{norm} and the side length of two nearest interacting cubes is b, then

$$\left(\frac{a}{\rho-a}\right)^{p+1} = \left(\frac{\frac{\sqrt{3}}{2}b}{2b-\frac{\sqrt{3}}{2}b}\right)^{p_{\mathrm{norm}}+1} \approx 0.7637^{p_{\mathrm{norm}}+1} \tag{26}$$

thus

$$p = 0.117(p_{\mathrm{norm}}+1)\Big/\log\left(\frac{\rho-a}{a}\right). \tag{27}$$

In the above equation, we have added 1 to p to ensure that its value is bigger than 0.

6. Numerical results

The adaptive algorithm has been implemented in a code written in C++ and tested with a 3D potential problem in a slender rectangular box with dimensions given by $a=2$, $b=2$ and $c=16$. We have considered two orientation cases of the box in the coordinate system, which are shown in Fig. 3. In the first case, the sides of the box are parallel/perpendicular to the coordinate axes, while in the second case they are not. For each case, the problem is solved in three ways: by methods with the oct-tree, by the binary tree and by the adaptive tree. In computations, we set the maximum number of boundary nodes in a leaf to be 60, and take both p in the standard algorithm and p_{norm} in the adaptive algorithm as 10. For GMRES, we terminate the iteration when the relative error is less than 10^{-5}. All computations are carried out on the same desktop computer with an Intel(R) Pentium(R) 4 CPU (1.99 GHz).

To assess the accuracy of the method, we calculate the relative error of nodal values of normal flux using the following 'global' L_2 norm:

$$err = \frac{1}{|q|_{\max}}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(q_i^{(\mathrm{e})}-q_i^{(\mathrm{n})})^2}, \tag{28}$$

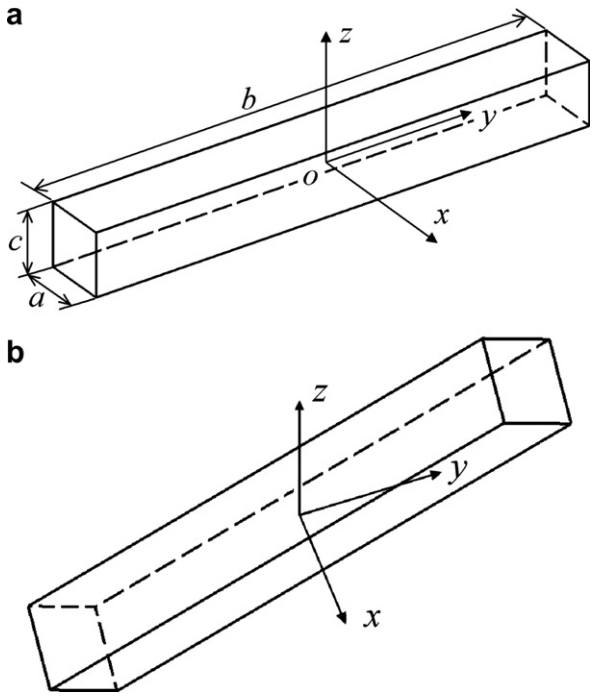

Fig. 3. A slender box with different orientations.

where q_i is the normal flux at node i, and $|q|_{\max}$ is the maximum value among the nodal values; n is the total number of nodes; the superscripts (e) and (n) refer to the exact and numerical solutions, respectively.

The following field distribution is used as the exact solution:

$$\phi = x^3 + y^3 + z^3 - 3yx^2 - 3xz^2 - 3zy^2. \tag{29}$$

Table 1
Results for the box parallel to the coordinate axes

Node distribution	DOFs	N_{Interact}	Depth	Leaves	T_{tree} (s)	T_{coef} (s)	T_{equ} (s)	err_q
Standard oct-tree								
[20, 20, 160]	13,600	24,024	7	440	1	63	146	5.28×10^{-3}
[28, 28, 224]	26,656	33,984	7	1120	3	139	260	4.10×10^{-3}
[40, 40, 320]	54,400	98,136	8	1912	6	227	773	3.32×10^{-3}
[49, 49, 392]	81,634	116,736	8	3392	30	495	968	2.03×10^{-3}
[56, 56, 453]	107,744	116,736	8	3392	35	765	1038	1.23×10^{-3}
Binary tree								
[20, 20, 160]	13,600	3766	11	288	2	94	51	5.34×10^{-3}
[28, 28, 224]	26,656	8262	13	682	4	167	126	4.17×10^{-3}
[40, 40, 320]	54,400	16,968	14	1131	8	548	277	3.40×10^{-3}
[49, 49, 392]	81,634	38,444	16	2268	34	629	607	2.11×10^{-3}
[56, 56, 453]	107,744	39,324	16	2450	38	1056	1560	1.16×10^{-3}
Adaptive tree								
[20, 20, 160]	13,600	4726	6	288	2	71	36	5.42×10^{-3}
[28, 28, 224]	26,656	15,084	7	678	4	111	86	4.34×10^{-3}
[40, 40, 320]	54,400	31,251	7	1134	9	375	183	3.65×10^{-3}
[49, 49, 392]	81,634	62,564	8	2268	37	465	361	2.66×10^{-3}
[56, 56, 453]	107,744	77,638	8	2452	40	659	476	1.74×10^{-3}

A Dirichlet problem is solved, where the essential boundary conditions are imposed on all the surfaces according to Eq. (29). For both geometries, we have performed computations on six node arrangements: [10, 80, 10], [20, 160, 20], [28, 224, 28], [40, 320, 40], [49, 392, 49] and [56, 453, 56], where $[n_a, n_b, n_c]$ refers to $n_b \times n_c$, $n_c \times n_a$ and $n_a \times n_b$ evenly spaced nodes on the surfaces perpendicular to the x, y and z axes, respectively.

The results obtained are summarized in Tables 1 and 2 for the two orientation cases of the box, respectively. In each table, the first and second columns of the table are the node arrangements and the total number of nodes; the third, fourth and fifth columns list the number of M2L translation (the total number of cell–cell interactions), the depth of the tree and the number of leaves. In the sixth, seventh and eighth columns, the total times used for constructing the tree, computing the near coefficients and solving the system equations are listed, respectively. The relative errors of nodal values for normal flux are presented in the ninth column.

The results show that, in all cases of node arrangement, the adaptive algorithm has achieved equivalent accuracy but higher efficiency. The binary tree indeed resulted in less M2L translations and in most cases runs faster than the oct-tree, but in one case runs even slower. The adaptive tree, however, outperformed both oct-tree and binary tree in all cases. It uses slightly more CPU time (in seconds) for constructing the tree. (Moreover, the time used for tree construction are so small that it can be ignored in the total computing time.) For computing near coefficients, the CPU seconds are very close to those used by the standard algorithm. However, for solving equations, the adaptive algorithm uses considerably fewer seconds, which is nearly one third of that used by the standard algorithm. The bottom rows of Tables 1 and 2 show that the new algorithm can solve a problem with 100 thousands nodes using about 20 min.

Tables 1 and 2 also show that the number of M2L translations in the adaptive tree is much bigger than in the binary tree, although the number of leaves of both trees are almost the same and the depth of the adaptive tree is much less shallow. This demonstrates that the reason for the better efficiency of the adaptive tree is not the reduction of M2L translation number, but the use of adaptive values of p for M2L translation. Plots of the M2L translation number against p for the boxes with sides parallel and nonparallel to the coordinate axes is presented in Fig. 4. It is obvious that the value of p needed by most of the M2L translations is merely 2.

Table 2
Results for the box nonparallel to the coordinate axes

Node distribution	DOFs	N_{Interact}	Depth	Leaves	T_{tree} (s)	T_{coef} (s)	T_{equ} (s)	err_q
Standard oct-tree								
[20, 20, 160]	13,600	36,688	6	614	1	97	262	5.13×10^{-3}
[28, 28, 224]	26,656	64,000	7	1276	3	151	550	4.01×10^{-3}
[40, 40, 320]	54,400	13,6912	7	2522	7	422	1385	3.56×10^{-3}
[49, 49, 392]	81,634	16,511	8	2884	30	723	2153	3.11×10^{-3}
[56, 56, 453]	107,744	16,674	8	3378	34	1110	3333	3.05×10^{-3}
Binary tree								
[20, 20, 160]	13,600	5922	11	333	2	101	70	5.05×10^{-3}
[28, 28, 224]	26,656	12,694	13	626	4	182	158	4.11×10^{-3}
[40, 40, 320]	54,400	26,454	14	1257	8	562	376	3.51×10^{-3}
[49, 49, 392]	81,634	46,410	15	2034	33	781	866	3.20×10^{-3}
[56, 56, 453]	107,744	55,920	15	2492	39	1060	1864	3.21×10^{-3}
Adaptive tree								
[20, 20, 160]	13,600	10,152	8	359	2	85	52	5.14×10^{-3}
[28, 28, 224]	26,656	31,475	9	705	4	137	124	4.26×10^{-3}
[40, 40, 320]	54,400	57,551	9	1395	9	371	270	3.69×10^{-3}
[49, 49, 392]	81,634	93,433	9	2229	35	552	519	3.99×10^{-3}
[56, 56, 453]	107,744	122,827	9	2731	40	762	679	3.98×10^{-3}

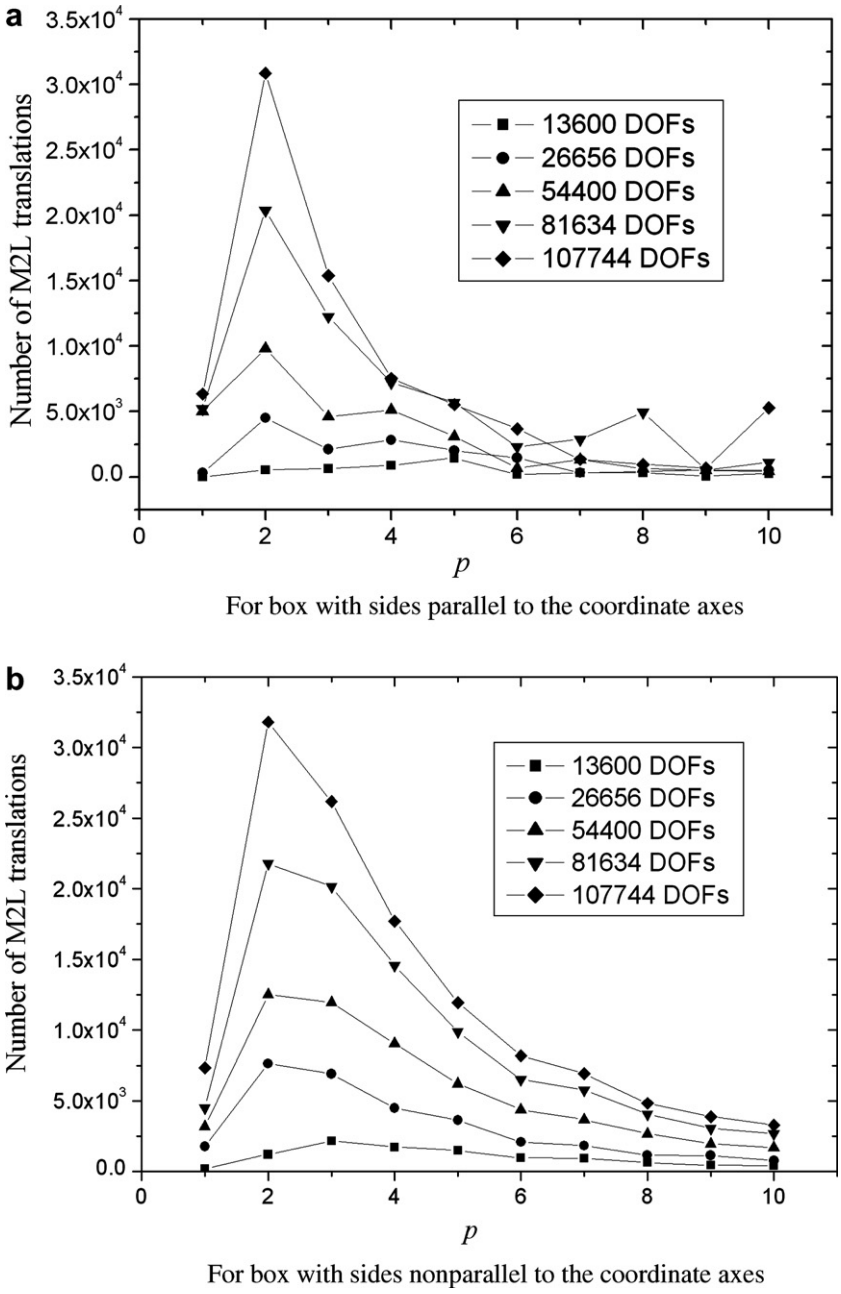

Fig. 4. Number of M2L translations against p.

7. Conclusion

In this paper, we have studied the performance of the FMM which uses a novel tree data structure, namely an adaptive tree. Instead of using cubes, the adaptive tree uses rectangular boxes to cluster boundary nodes into groups, and the boxes are split according their shapes in the process of constructing the tree. Therefore, the new tree is more flexible in matching the geometry (global and local) of the computational domain. Most importantly, the number of terms of the truncated series for M2L translations is determined by the distance between the two interaction boxes. The adaptive algorithm can provide a huge improvement in efficiency over the standard oct-tree.

A numerical example is presented to study the performance of the proposed algorithm. Results obtained show that the adaptive algorithm leads to trees with more compact cells and runs significantly faster than the standard oct-tree, and also outperforms the algorithm with binary tree.

Acknowledgments

This work is supported by the national 973 program (China) under the grant number 2004CB719402 and the program for New Century Excellent Talents in University (NCET-04-0766).

References

[1] V. Rokhlin, Rapid solution of integral equations of classical potential theory, J. Comput. Phys. 60 (1985) 187–207.
[2] L. Greengard, V. Rokhlin, A new version of the Fast Multipole Method for the Laplace equation in three dimensions, Acta Numer. 6 (1997) 229–269.
[3] R.J. Anderson, Tree data structures for N-body simulation, SIAM J. Comput. 28 (1999) 1923–1940.
[4] M. Urago, T. Koyama, K. Takahashi, S. Saito, Y. Mochimaru, Fast multipole boundary element method using the binary tree structure with tight bounds: application to a calculation of an electrostatic force for the manipulation of a metal micro particle, Eng. Anal. Boundary Elements 27 (2003) 835–844.
[5] C.A. White, M. Head-Gordon, Fractional tiers in fast multipole method calculations, Chem. Phys. Lett. 257 (1996) 647–650.
[6] J.M. Zhang, Z.H. Yao, H. Li, A hybrid boundary node method, Int. J. Numer. Meth. Eng. 53 (2002) 751–763.
[7] J.M. Zhang, Masa. Tanaka, T. Matsumoto, Meshless analysis of potential problems in three dimensions with the hybrid boundary node method, Int. J. Numer. Meth. Eng. 59 (2004) 1147–1160.
[8] J.M. Zhang, Masa. Tanaka, M. Endo, The hybrid boundary node method accelerated by fast multipole method for 3D potential problems, Int. J. Numer. Meth. Eng. 63 (2005) 660–680.
[9] K. Yoshida, Applications of fast multipole method to boundary integral equation method, Ph.D. Dissertation, Department of Global Environment Engineering, Kyoto University, 2001.
[10] H.G. Petersen, D. Solvason, J.W. Perram, The very fast multipole method, J. Chem. Phys. 101 (1994) 8870–8876.
[11] D. Solvason, H.G. Petersen, Error estimates for the fast multipole method, J. Stat. Phys. 86 (1997) 391–420.
[12] H.G. Petersen, E.R. Smith, D. Solvason, Error estimates for the fast multipole method. II. The three-dimensional case, Proc. R. Soc. Lond. A 448 (1995) 401–418.