Transactions of JASCOME, Vol. 6, No. 2 (Dec. 2006), Paper No. 10-061208 **JASCOME**

TREE DATA STRUCTURES FOR FAST MULTIPOLE METHOD

Jianming ZHANG [1)], Masataka TANAKA [2)]

1) Faculty of Engineering, Shinshu University, (Nagano 380-8553, e-mail: zhangjm@homer.shinshu-u.ac.jp)

2) Faculty of Engineering, Shinshu University, (Nagano 380-8553, e-mail: dtanaka@gipwc.shinshu-u.ac.jp)

In this paper, we study the tree data structures for Fast Multipole Method. We have considered three options, namely the degree of the decomposition, the clustering cell boundary and the number of expansion terms in *multipole to local* translations. These options yield twelve separate algorithms. Computational study on domains of various geometrical shapes reveals that the binary tree is sometimes better sometimes worse than the standard oct-tree, while the adaptive tree with tight bounds and adaptive value of the number of expansion terms exhibits the best performance in all cases.

Keywords: fast multipole method; tree data structure; oct-tree; binary tree; adaptive tree; boundary element method; hybrid boundary node method

1. Introduction

The Fast Multipole Method (FMM) of Roklin and Greengard [1] 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).

The FMM 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 terms of series) to approximate the effects of a distant group of particles, namely elements in Boundary Element Method (BEM) or nodes in Hybrid Boundary Node method (HdBNM) [2], on a local group, and translations (M2M, M2L and L2L) 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 [3] proposed a 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 [4] 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 [5] implemented this idea in a FMM simulation of electrostatic field, and gained an efficiency improvement of two to three times over the standard algorithm. The authors of this paper have also proposed a new adaptive node-cluster algorithm [6], in which rectangular boxes are used instead of cubes, and the boxes are subdivided according to their shapes. More importantly, the values of p for the M2L translations are determined by the distance between the two interaction boxes. Numerical examples have shown that the new adaptive algorithm can provide a huge improvement in efficiency over the standard oct-tree.

The present paper attempts to give a direct comparison between different tree data structures. Our main contribution consists in a systematic computational study aimed at assessing the effectiveness of the tree data structures for problems with different domain geometries. We have considered three options that can be applied independently, yielding twelve separate algorithms. The first option is the degree of the decomposition. There are three choices, namely the standard oct-tree, the binary tree and the adaptive tree for this option. The second option relates to the definition of cell boundary. The choices are either loose bounds or tight bounds. The third option concerns the values of p for the M2L translations. We can use either a fixed value of p for all M2L translations or an adaptive value for different translations. The comparison has demonstrated that effectiveness of a tree data structure depends on the shape of the computational domain it is applied to. For some domain geometries the binary tree is superior to the oct-tree, but it may be inferior for other geometries. The adaptive tree with tight bounds and adaptive value of p, however, is the best algorithm in all cases.

2. Review of the fast multipole method

In either BEM or HdBNM, 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 k-th step, the matrix-vector multiplication at the k+1-th step is

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

or

$$x_I'^{k+1} = \sum_{J=1}^{N} \int_{\Gamma_I} \frac{\partial \phi_J^s}{\partial n} v_I x_J^k \, d\Gamma \qquad (2)$$

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; N is the total number of nodes or elements; v_I is a weight function or a shape function; Γ_I is a local region around a node $\mathbf{s}_I$ or the I-th element; ϕ_I^s is the fundamental solution with the source at a node $\mathbf{s}_I$. For 3-D potential problems, the fundamental solution can be written as

$$\phi_I^s = \frac{1}{4\pi r(Q,\mathbf{s}_I)} \qquad (3)$$

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

Direct computation of Eqs. (1) and (2) gives an $O(N^2)$ algorithm. The FMM can be employed to reduce the complexity to $O(N)$.

2.1 Cell-cell interaction

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 [7]

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

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

Here (r,α,β) are 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

$$\frac{1}{|\mathbf{r}_1-\mathbf{r}_2|} = \begin{cases} \sum_{n=0}^{\infty}\sum_{m=-n}^{n} R_n^m(\mathbf{r}_1)\overline{S_n^m(\mathbf{r}_2)}, & |\mathbf{r}_1|<|\mathbf{r}_2| \\ \sum_{n=0}^{\infty}\sum_{m=-n}^{n} R_n^m(\mathbf{r}_2)\overline{S_n^m(\mathbf{r}_1)}, & |\mathbf{r}_1|>|\mathbf{r}_2| \end{cases} \qquad (4)$$

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

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

$$S_n^m(\mathbf{r}_1-\mathbf{r}_2) = \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'} \overline{R_{n'}^{m'}(\mathbf{r}_2)} S_{n+n'}^{m+m'}(\mathbf{r}_1) \qquad (5)$$

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

$$R_n^m(\mathbf{r}_1-\mathbf{r}_2) = \sum_{n'=0}^{n}\sum_{m'=-n'}^{n'} R_{n'}^{m'}(-\mathbf{r}_2) R_{n-n'}^{m-m'}(\mathbf{r}_1) \qquad (6)$$

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 $O(N_a \times N_b)$ (Figure 1a). In the cell-cell strategy, it is reduced to $O(N_a + N_b)$ (Figure 1b).

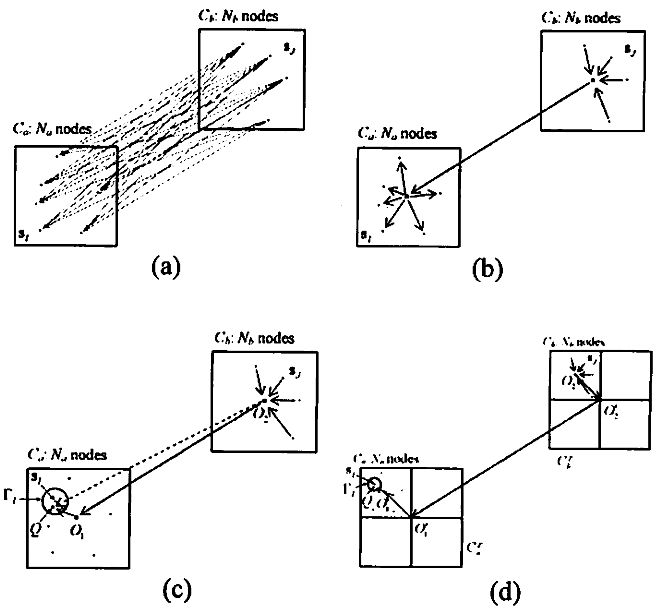

Figure 1. Interaction between two cells.

Substituting Eq. (3) into Eq. (1) and using the first addition theorem, with the summation over the nodes included in C_b (see Figure 1c), we obtain

$$\begin{aligned} &\sum_{J=1}^{N_b} \int_{\Gamma_I} \phi_J^s v_I(Q) x_J^k \, d\Gamma \\ &= \sum_{n=0}^{\infty}\sum_{m=-n}^{n} \int_{\Gamma_I} \frac{1}{4\pi} \overline{S_n^m(\overline{O_2Q})} v_I(Q) \, d\Gamma \, M_n^m(O_2) \end{aligned} \qquad (7)$$

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

$$M_n^m(O_2) = \sum_{J=1}^{N_b} R_n^m(\overline{O_2\mathbf{s}_J}) x_J^k \qquad (8)$$

Using further the second addition theorem, Eq. (7) becomes

$$\begin{aligned} &\sum_{J=1}^{N_b} \int_{\Gamma_I} \phi_J^s v_I(Q) x_J^k \, d\Gamma \\ &= \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'} \int_{\Gamma_I} \frac{1}{4\pi} R_{n'}^{m'}(\overline{O_1Q}) v_I(Q) \, d\Gamma \, L_{n'}^{m'}(O_1) \end{aligned} \qquad (9)$$

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

$$L_{n'}^{m'}(O_1) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} (-1)^{n'} \overline{S_{n+n'}^{m+m'}(\overline{O_1O_2})} M_n^m(O_2) \qquad (10)$$

Equation (10) 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 Figure 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. (10) becomes

$$L_{n'}^{m'}(O_1) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} R_{n-n'}^{m-m'}(\overline{O_1'O_1}) L_n^m(Q_1') \tag{11}$$

and

$$L_n^m(O_1') = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (-1)^n \overline{S_{n+n'}^{m+m'}}(\overline{O_1'O_2'}) M_{n'}^{m'}(Q_2') \tag{12}$$

$$M_{n'}^{m'}(Q_2') = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} R_n^m(\overline{O_2'O_2}) M_{n-n'}^{m-m'}(Q_2) \tag{13}$$

The above process can be recursively repeated down to the root cell that 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 the pair of points in the solid spherical harmonics, so that the *coefficients of multipole expansion* and *local expansion* are related only to the 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.

2.2 Tree construction and FMM algorithm

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 resulting 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.

The standard FMM algorithm uses an oct-tree. The entire computational domain is assumed to lie inside a cube, which is referred to 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 (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. (1).

3. Comparison schemes

Although the FMM possesses linear asymptotic performance, the constant factor of the run time still remains very large. This is mainly due to the large number of M2L translations for each cube at every level. One way to reduce the number of M2L translations is to find a better tree data structure [4, 5]. 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 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. 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. In this section, we explore how to improve the constant factor of the algorithm by improving the tree data structure. We try to give a collection of results that show how different tree properties influence the performance of algorithms, and to determine the best data structure to use in the FMM. For the purpose, we consider three options that can be applied independently. The first option has three choices, while the second and the third have two choices, respectively. Thus, these options will totally yield twelve algorithms.

3.1 Decomposition strategies

The first option is the degree of the decomposition. Here the degree means the number child cells of a parent cell in the decomposition. There are three choices for this option. The first choice is the standard oct-tree decomposition, in which each cell is equally subdivided into eight child cells. We have described this tree in detail in the previous section. The second choice is a binary decomposition which splits the longest dimension of a cell evenly. The binary tree is now becoming popular in the FMM community. The third choice is an adaptive decomposition we have proposed. The adaptive decomposition is based on the standard oct-tree decomposition, but differs in the following aspects:

1. Instead of using cubes, the adaptive decompostion uses rectangular boxes to decompose the computational domain. It is believed that a rectangular box is more flexible in matching structures.

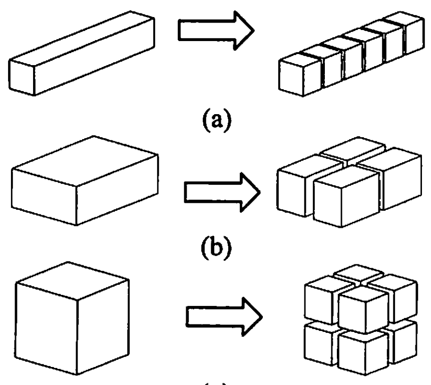

Figure 2. Subdivision of a box.

2. In contrast to the oct-tree decomposition, where subdivision of a cube is oblivious to geometry, a box in the adaptive decomposition is split into child boxes based on its shape. Let L_1, L_2, and L_3 be the side lengths of a box in the three coordinate directions, respectively, and suppose $L_1 > L_2 > L_3$ without loss of generality. 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=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.

Therefore, the degree of the adaptive decomposition is neither 8 nor 2. The number of offspring of a parent box is dependent on the shape of the box.

3.2 Cell bounds

The second option relates to the definition of cell boundary. The choices are either loose bounds or tight bounds. Boxes with tightened bounds have been investigated by Anderson [4], and are proven to be more efficient in separating a node set into well-separated clusters. One method for choosing the rectangular boxes is to subdivide the cubes when the cells are split. The bounds of the cells are inherited from the bounds of the parent cell. This method is referred to as *loose bounds* and is the way that used in the standard oct-tree decomposition. An alternative way to define the cell boundary is to use a smallest box to enclose the cluster of boundary nodes. The bounds are actually the bounds of the node set and are 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.

3.3 Number of terms in truncated M2L translation series

In practical computation, the infinite series in Eqs. (9-13) are truncated after p terms. The error estimation of the truncated series can be found in [3]. The third option is concerned with how to choose the value of p. The way in the standard FMM algorithm is to use a fixed p for all M2L translations. Alternatively, we can also use adaptive values of p of the truncated series for different M2L translations. In this study, the adaptive value is determined by

$$p = 0.117 p_{norm} \Big/ \log(\frac{a}{\rho - a}) \qquad (14)$$

where p_{norm} is a norm value; a is the maximum radius of the two spheres that enclose the two interacting cells, and ρ is the distance between the centers of the two boxes.

4. Numerical comparative results

The three options of tree data structure result in twelve algorithms. These algorithms have been implemented in computer codes written in C++. We will test and compare these algorithms on two objects: a bulky object namely a cube and a slender rectangular box, which are shown in Fig. 3 and Fig. 4, respectively. The sides of the cube are parallel/perpendicular to the coordinate axes, while for the slender box they are not. The dimensions are given by $a = 2$, $b = 16$ and $c = 2$.

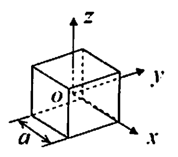

Figure 3.The cube.

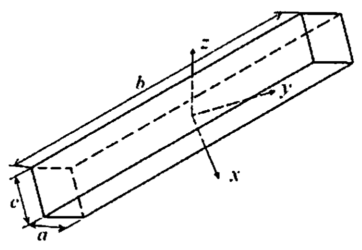

Figure 4. The slender box.

We consider the following field distribution:

$$\phi = x^3 + y^3 + z^3 - 3yx^2 - 3xz^2 - 3zy^2 \qquad (15)$$

and solve a Dirichlet problem, where the essential boundary conditions are imposed on all the surfaces according to Eq. (16). We have performed computations on four node arrangements: $[40, 40, 40]$, $[80, 80, 80]$, $[120, 120, 120]$ and $[160, 160, 160]$ for the potential problem in the cube, and five node arrangements: $[20, 160, 20]$, $[28, 224, 28]$, $[40, 320, 40]$, $[49, 392, 49]$ and $[56, 453, 56]$ for the problem in the slender box, 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 with side lengths b and c, c and a, and a and b, respectively.

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

$$err = \frac{1}{|q|_{\max}}\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(q_i^{(e)} - q_i^{(n)}\right)^2} \qquad (16)$$

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.

In the computations, we set the maximum number of boundary nodes in a leaf to be 60, and take both the fixed p and the normal value of the adaptive p, p_{norm} as 10. The preconditioned GMRES is employed with the preconditioner being the inverse of the blocked diagonal matrix corresponding to the nodes in leaves. We terminate the iteration of GMRES 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 (1.99GHz).

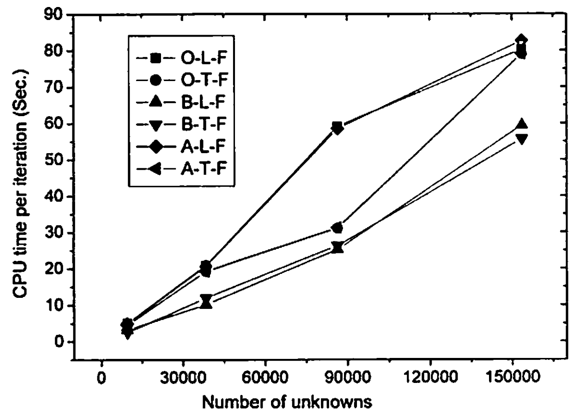

Figure 5. CPU times for the cube by algorithms with fixed p.

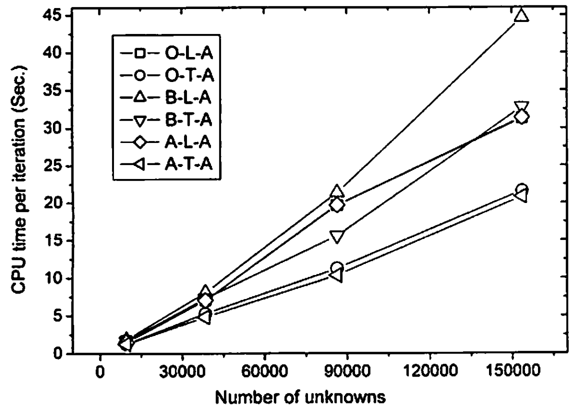

Figure 6. CPU times for the cube by algorithms with adaptive p.

Our performance measure is the CPU time used for one iteration. Figs. 5-8 plot the CPU seconds as a function of the number of unknowns for the two geometries by the twelve algorithms. In the figures, the first letters of the legends, O, B and A denote oct-tree, binary tree and adaptive tree, respectively. The second letters L and T stand for loose cell bounds and tight bounds; and the third letters F and A for fixed and adaptive p, respectively. From the figures, we make the following observations:

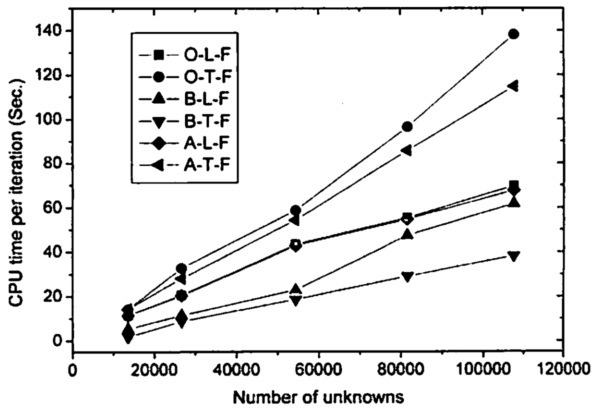

Figure 7. CPU times for the slender box by algorithms with fixed p.

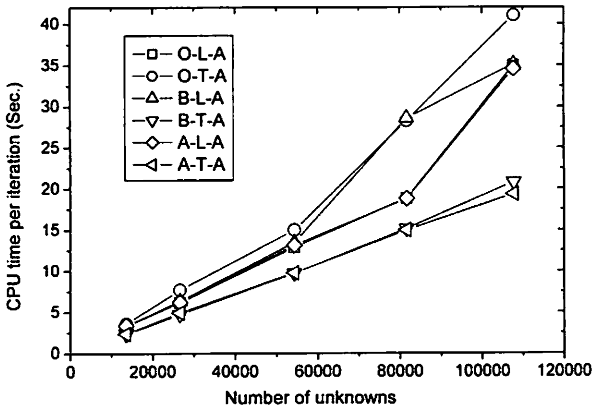

Figure 8. CPU times for the slender box by algorithms with adaptive p.

1. Comparing Figs. 5 and 6, and Figs. 7 and 8, we see that algorithms with adaptive p are asymptotically superior to that with fixed p. The use of adaptive p improves the computational efficiency by nearly two folds. With fixed p, the binary tree with tight cell bounds (B-T-F) provides the best efficiency (see Figs. 5 and 7), while with adaptive p, the adaptive tree with tight cell bounds (A-T-A) is the best algorithm (see Figs. 6 and 8).
2. In the case of the cube (see Fig. 6), the oct-tree with tight cell bounds and adaptive p (O-T-A), has very close computational efficiency as the adaptive tree with tight cell bounds and adaptive p (A-T-A). In this case, the oct-tree is superior to the binary tree (B-T-A). However, it is inferior to the binary tree (B-T-A) for the slender box (see Fig. 8). This implies that the effectiveness of a tree data structure depends on the shape of the computational domain it is applied to. We happily see that the adaptive tree with tight cell bounds and adaptive p (A-T-A) outperforms all other algorithms for both two computational domains.
3. Figs. 5 and 6 show that, for problem in the cube, O-T-F, O-T-A, B-T-F and B-T-A perform better than O-L-F, O-L-A, B-L-F and B-L-A, respectively. In the case of the slender box (see Figs. 7 and 8), B-T-F and B-T-A perform better than B-L-F and B-L-A, while O-T-F and O-T-A worse than O-L-F and O-L-A, respectively. This demonstrates that the effectiveness of using tight cell bounds depends not only on the shape of the

computational domain, but also on the choice of data tree. For binary tree, tight cell bounds can provide better efficiency. For oct-tree however, it may hurt the performance. This observation is consistent with that in [4], where the author showed that, in two dimension case, binary trees with tight bounds have the same asymptotic run time as the trees with loose bounds. However, for quad-trees, tight bounds can lead to trees that are much worse than trees with loose bounds.

In order to check if the above observations are valid in even larger computational scales, we have also performed computations on the node arrangement [200, 200, 200] for the potential problem in the cube. The total number of unknowns of this computation is 240000. The results are summarized in Table 1. The first column lists the algorithms used; the second column lists the number of M2L translations (the total number of cell-cell interactions).The third, fourth and fifth columns list the times used for constructing the tree, solving the system of equations and one iteration are listed, respectively. The relative errors of nodal values for normal flux are presented in the sixth column. The results show that algorithms have similar accuracy, however, the binary trees are considerably worse than other trees, although they do decrease the number of M2L translations. The best algorithm is O-T-A, which perform slightly better than A-T-A. The reason for the very bad performance of the binary tree in this case has not been clear and needs further investigation. It is also seen that the CPU times for constructing the trees are very close for all the algorithms. (Moreover, these times are so small that they can be ignored when compared with that used for solving the system equation.)

Table 1. Results of computations on node arrangement [200, 200, 200] for the cube.

Algor.	$N_{Interact}$	T_{tree} (sec.)	T_{equ} (sec.)	T_{iter} (sec.)	err_q ($\times 10^{-4}$)
O-L-F	340328	195	2603	100	3.35
O-T-F	263114	197	3196	94	2.94
B-L-F	195648	199	42003	1167	2.77
B-T-F	130614	194	26918	728	2.50
A-L-F	249896	196	2665	103	3.34
A-T-F	141318	199	3309	97	2.94
O-L-A	340328	198	25462	979	3.52
O-T-A	263114	197	1120	33	3.55
B-L-A	195648	199	33058	918	2.81
B-T-A	130614	196	36450	985	2.53
A-L-A	249896	197	26234	1009	3.52
A-T-A	141318	199	1252	34	3.10

5. Conclusions

In this paper, we have performed a comparative study on the tree data structure for FMM. We have considered twelve algorithms that come from three options, namely the decomposition strategy, definition of cell bounds and the determination of number of terms in truncated M2L translation series. The numerical results have demonstrated that the effectiveness of a tree data structure depends on the shape of the computational domain it is applied to. The binary tree is sometimes better and sometimes worse than the standard oct-tree. The tight cell bounds can be used for binary and the adaptive trees without hurting the performance, while for oct-tree the tight cell bounds may cause a substantial slowdown. The argument of this paper is that the adaptive tree with tight cell bounds and adaptive number of terms in truncated M2L translation series is the best algorithm for FMM.

Acknowledgements

The support of *Japan Society for the Promotion of Sciences* (JSPS) and Shinshu University, Nagano, Japan, is gratefully acknowledged. Parts of this work were financially supported by the Grant-in-Aid for JSPS Fellows No. 17_05075.

References

1. Rokhlin V., Rapid solution of integral equations of classical potential theory. *J. Comput. Phys.*, Vol. 60 (1985), pp. 187-207.

2. Zhang J.M., Yao Z.H., Li H., A hybrid boundary node method. *Int. J. Num. Meth. Engng.*, Vol, 53 (2002), pp. 751-763.

3. Greengard L., Rokhlin V., A new version of the Fast Multipole Method for the Laplace equation in three dimensions. *Acta Numerica*, Vol. 6 (1997), pp. 229-269.

4. Anderson R.J., Tree data structures for N-body simulation. *SIAM J. Comput.*, Vol. 28 (1999), pp. 1923-1940.

5. Urago M., et al., 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. *Engineering Analysis with Boundary Elements*, Vol. 27 (2003), pp. 835-844.

6. Zhang J.M., Tanaka Masa., An effective tree data structure in Fast Multipole Method, *Transactions of the Japan Society for Computational Methods in Engineering*, Vol. 6 (2006), pp. 17-22.

7. Yoshida K., Applications of Fast Multipole Method to Boundary Integral Equation Method, Ph.D. dissertation, Department of Global Environment Engineering, Kyoto University, 2001.