Journal of Applied Mechanics Vol.7,pp.287-294 (August 2004) JSCE

Parallelization of Fast Multipole Accelerated BIEM for SMP Computers

共有メモリ型コンピュータにおける高速多重極積分方程式法の並列化

Hidenori MUNAKATA*, Naoshi NISHIMURA**
宗像秀紀・西村直志

*Graduate Student, Department of Civil and Earth Resources Engineering, Kyoto University, Kyoto 606-8501
**Mem. JSCE, Dr. Eng., Prof., Academic Center for Computing and Media Studies, Kyoto University, Kyoto 606-8501

A parallel fast multipole accelerated boundary integral algorithm for solving boundary value problems is presented in this paper. This paper focuses on the downward pass, the upward pass and the preconditioning process, which are considered to be the expensive parts in the FMM(Fast Multipole Method) algorithm. All of them are parallelized with OpenMP. The performance of the parallel fast multipole method is tested with two numerical examples, namely the crack problem for Laplace's equation in 2-D and the elastostatic inclusion problem in 3-D. From those numerical examples, we can say that the speedup achieved with OpenMP is satisfactory in spite of the relatively small amount of efforts for programmers and small change in the program structure. Particularly remarkable is the fact that the effect of this parallelization in the fast multipole method is more pronounced in large scale problems or in 3-D problems.

Key Words : *BIEM, fast multipole method, parallelization, SMP*

1. Introduction

Suppose that one introduces N unknowns to discretize a boundary integral equation. Conventional BIEMs(Boundary Integral Equation Method) will produce an $N \times N$ dense and nonsymmetric matrix and this linear system will be solved with either direct or iterative solvers. This process undoubtedly requires operations of the complexity proportional to N^3 with direct solvers and to N^2 with iterative solvers. Also, the memory requirement is proportional to N^2. BIEM is therefore considered quite expensive for large problems, compared to other major numerical methods such as FDM or FEM, which need only $O(N)$ operations and memory for an equivalent task, because their coefficient matrices are banded. This fact made BIEM a loser for large problems[1]. However, recent developments of fast BIEM have brought BIEM back to large scale problems, showing that BIEM can solve problems with $O(N)$ operations with the help of FMM.

FMM was introduced by Rokhlin[2] for solving an integral equation for Laplace's equation in 2-D. FMM was further developed and made famous by Greengard[3] as he applied FMM to many body problems.

The efficiency of the Fast multipole accelerated BIEM is achieved as one evaluates the contributions of far elements at once using the multipole expansion. This is in contrast with the conventional BIEM, in which we evaluate the contributions of all the elements one by one. The performance of the Fast multipole accelerated BIEM

on single CPU machines has been tested by many investigators (See Nishimura[1] for references). If one seeks to further enhance the efficiency and applicability of this method in larger problems, one will have to investigate parallelization of the code.

Parallel computers can be classified into two categories, i.e. distributed memory and shared memory machines. A popular architecture of the latter type is the SMP (Symmetric MultiProcessors). In SMP computers, the parallelization is usually loop based, i.e., loops in a serial program is divided into threads and each processor will execute one thread of each loop. The task of programmers for SMP type computers is then to write directives with OpenMP[4] in front of the "do loop". The advantage of this type of parallelization is that the programmer needs little effort to make changes in the program structure

¶ Dedicated to the memory of Prof. Michihiro KITAHARA

and algorithm to convert serial programs to parallel ones. On the other hand, distributed memory systems demand the programmer to specify the distribution of the work load and communication among nodes with MPI[5], etc. On distributed memory systems, the structure based parallel computation, which is generally not very easy, is known to be more efficient than the loop based one[6].

In spite of such an advantage of SMP computers, researches on SMP computers have been scarce compared to those of distributed memory machines, because large systems of SMP computers have been rather rare until recently. Actually, the parallelization of BIEM and FMM on distributed memory systems has been well studied and a remarkable speedup has been achieved [7], but researches on SMP computers have been limited only to conventional BIEM (see Nishimura *et al*[8] for attempts of a hybrid parallelization on SMP computers).

But, recently, the situation has changed. For example, the Academic Center for Computing and Media Studies (ACCMS) of Kyoto University has replaced its vector parallel computer Fujitsu VPP800 with a scalar parallel computer HPC2500. The HPC2500 of ACCMS consists of 11 SMP nodes, each of which has 128 CPUs and 512 GB of shared memory. This trend may continue, considering the fact that large scalar SMP computers are usually made of less expensive components than vector parallel machines, and the importance of parallel algorithms for SMP computers may increase. This paper, therefore, focuses on the parallelization of FMM on SMP computers and tests the performances of the parallel codes via numerical examples in two dimensional crack problems for Laplace's equation and in three dimensional elastostatic inclusion problems.

2. Formulation

In this section we shall briefly describe the FMM formulation for two dimensional crack problems for Laplace's equation for simplicity. See Nishimura [1] for the FMM formulations in three dimensional elastostatics.

2.1 Crack Problems for Laplace's Equation in 2-D

(1) Integral representation

Let S be a set of cracks in R^2 and D be $R^2 \backslash S$. The governing equations for the crack problem for two dimensional Laplace's equation are given as:

$$u_{,ii} = 0 \quad \text{in} \quad D \tag{1}$$

$$\frac{\partial u}{\partial n} = 0 \quad \text{on} \quad S \tag{2}$$

$$u(x) \to u_\infty(x) \quad \text{as} \quad |x| \to \infty \tag{3}$$

$$\phi(x) = u^+ - u^- = 0 \quad \text{on} \quad \partial S \tag{4}$$

where u_∞ is a harmonic function in R^2 (the solution when no cracks are considered), n denotes the unit normal to S and ϕ denotes the crack opening displacement obtained from the following integral equation:

$$⨍_S \frac{\partial^2 G(x-y)}{\partial n_x \partial n_y} \phi(y)\, \mathrm{d}S_y = -\frac{\partial}{\partial n} u_\infty(x) \qquad x \in S. \tag{5}$$

In (5), G denotes the fundamental solution of the 2 dimensional Laplace's equation given by

$$G(x) = -\frac{1}{2\pi} \log|x| \tag{6}$$

and $⨍$ denotes the finite part of a divergent integral.

(2) Multipole expansion

Evaluation of the L.H.S.(Left Hand Side) of the hypersingular equation is possible if one can calculate the following integral:

$$\left(\frac{\partial}{\partial x_1} - \mathrm{i}\frac{\partial}{\partial x_2}\right) \int_S \frac{\partial G(x-y)}{\partial n_y} \phi(y) \mathrm{d}S_y \tag{7}$$

Putting $z = x_1 + \mathrm{i}x_2$ and $\zeta = y_1 + \mathrm{i}y_2$, one further simplifies the integral into the following form:

$$\frac{1}{2\pi \mathrm{i}} \int_S \frac{\phi(\zeta)}{(z-\zeta)^2} \mathrm{d}\zeta \tag{8}$$

This integral is evaluated in terms of the multipole expansion. To see this, we set:

$$I_p(z) = \frac{z^p}{p!}, \quad O_p(z) = p! z^{-1-p} \tag{9}$$

Assuming $|z - z_0| < |\zeta - z_0|$ we have:

$$(8) = \sum_{q=0}^{\infty} (-1)^q I_q(z - z_0) \sum_{p=q}^{\infty} O_{p+1}(z_0 - \zeta_0) M_{p-q}(\zeta_0) \tag{10}$$

where z_0 is a point near z, ζ_0 is a point near S and $M_q(\zeta_0)$ is the multipole moment centered at ζ_0 defined by:

$$M_q(\zeta_0) = \frac{1}{2\pi \mathrm{i}} \int_S I_q(\zeta - \zeta_0) \phi(\zeta) \mathrm{d}\zeta \tag{11}$$

The center of the multipole moment is shifted with the M2M formula given by:

$$M_q(\zeta_1) = \sum_{r=0}^{q} I_{q-r}(\zeta_0 - \zeta_1) M_r(\zeta_0) \tag{12}$$

The integral in (8) can be rewritten in terms of the local expansion as follows:

$$\sum_{q=0}^{\infty} I_q(z - z_0) L_q(z_0) \tag{13}$$

where $L_q(z_0)$ is the coefficient of the local expansion centered at z_0 given by:

$$L_q(z_0) = \sum_{p=q}^{\infty} (-1)^q O_{p+1}(z_0 - \zeta_0) M_{p-q}(\zeta_0) \tag{14}$$

which we call the M2L formula. Finally, the center of the local expansion is shifted with the L2L formula given by:

$$L_r(z_1) = \sum_{p \geq r} L_p(z_0) I_{p-r}(z_1 - z_0) \qquad (15)$$

3. Fast Multipole Algorithm

3.1 A Hierarchical Structure

The tree structure in 2-D is described here. The 3-D counterpart is given just by replacing the expression "quad-tree" with "oct-tree".

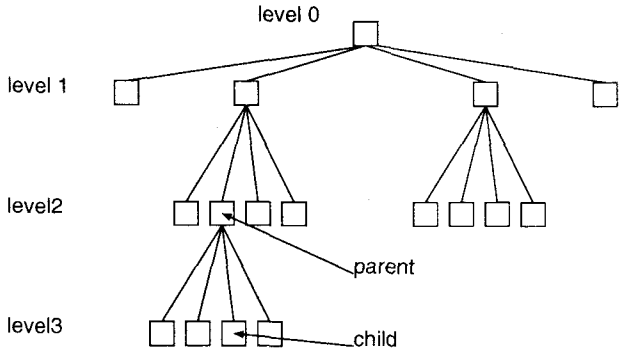

Fig. 1 The quad tree structure.

The whole boundary is divided into many cells as shown in Fig.2. Also shown in Fig.1 is the hierarchical quad-tree structure. As shown in these figures,

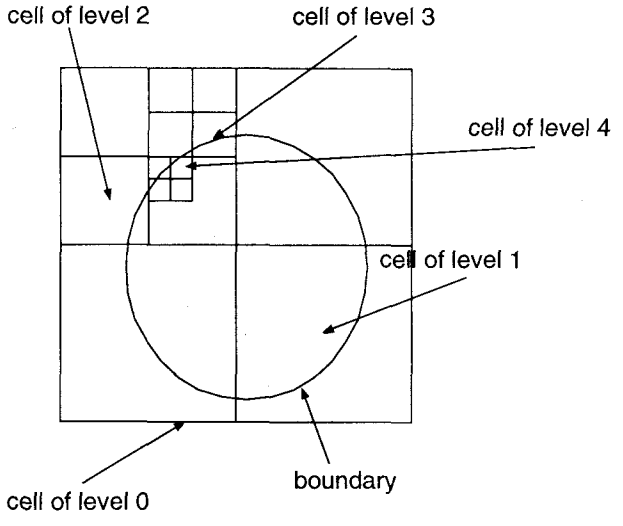

Fig. 2 Cells on the boundary.

each cell is divided to four subcells (children) until the maximum number of elements in the cell becomes less than a preset value N_0, or the level of the cell reaches a preset maximum level. The parameter N_0 is set empirically between 50 and 500 so as to best save the memory and computational cost. A childless cell is termed "leaf". The relationships among cells are shown in Fig.3. For the cell painted in black, those cells which are next to it are termed "neighboring cells", those cells whose parents are next to its parent are termed "cells in the interaction list' and other cells are termed "far cells".

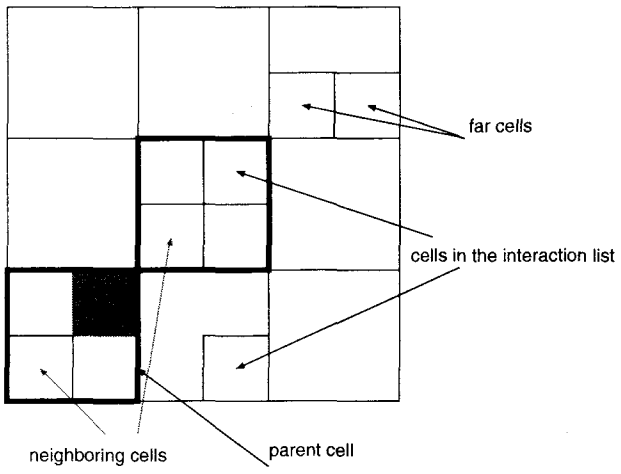

Fig. 3 Relationship among cells.

3.2 Main Parts of the Fast Multipole Algorithm

(1) Upward pass

The fast multipole algorithm is described here. For every leaf, the multipole moments around the cell center are calculated. The center of the multipole moment is then shifted to that of the parent cell (M2M). The multipole moments of the parent are obtained as the sum of the moments of its children. In this way, from the bottom level to level 2, the multipole moments around the center of each cell are obtained.

(2) Downward pass

On level 2, the coefficients of the local expansion are obtained from the multipole moments of cells in the interaction list (M2L). Children get the coefficients of the local expansions from their parents with the shift of the centers (L2L) and get the coefficients of the local expansions from the cells in the interaction lists (M2L) whose contributions are not included in the coefficients of the local expansions of their parents. On every leaf, Ax (the discretized L.H.S. of eq.(5)) is obtained with the direct calculation of the contributions of the elements in neigboring cells and the leaf itself as in the conventional BIEMs and from the coefficients of the local expansion of the leaf.

Fortran-like pseudocodes for the upward and downward passes go as follows:

C UPWARD PASS

```
DO level=maxlevel, 2, −1

    DO cell = all the cells of the current level

        if (cell is a leaf) then
            calculate M(cell)
        endif

        if (level ≠ 2) then
            add M(cell) to M(parent_cell) after
            the shift of the center (M2M)
        endif

    ENDDO

ENDDO
```

```
C DOWNWARD PASS

DO level=2, maxlevel

    DO icell = all the cells of the current level

        if (level ≠ 2) then
            get L(icell) from L(parent_icell)
            after the shift of the center (L2L)
        endif

        DO jcell = all the cells in the interaction
                   list of icell

            add M(jcell) to L(icell) with M2L
        ENDDO

        DO jcell = all the cells in the neighborhood
                   of icell

            if (either icell or jcell is a leaf) then
                evaluate Ax with the direct
                method
            endif

        ENDDO

        if (icell is a leaf) then
            evaluate Ax from L(icell)
        endif

    ENDDO

ENDDO
```

4. Parallel Fast Multipole Algorithm

4.1 Parallelization Strategy

It is seen that the sum of the elapsed times for the upward pass, downward pass and preconditioning gives almost all the elapsed time for solving the boundary value problem. Indeed, a numerical experiment of a three dimensional elastostatic problem is carried out to check this and the result is shown in Table 1. See section 5.2 for the detail of the analysis.

Table 1 The elapsed time for CNT analysis (175104DOF)

D(s)	U(s)	P(s)	total time(s)
6711.21	211.68	33.84	6962.13

D: downward process
U: upward process
P: preconditioning process

In this example 0.486% of the elapsed time is consumed in the preconditioning process, 3.040% is in the upward pass and 96.3960% is in the downward pass, the sum of which gives 99.9% of the whole time. Other processes such as making tree structure or normalization of the Krylov bases in GMRES (generalized minimal residual method) are negligible compared to those three processes. Therefore it is considered satisfactory to parallelize only these three processes.

4.2 Automatic Parallelization

The HPC2500 in ACCMS is implemented with the automatic parallelization as a compiler option. Automatic parallelization is tried first, because if it works well we do not need to make efforts for parallelization manually. With the automatic parallelization, however, we found that the calculations in GMRES are parallelized, but none of the three processes mentioned above are parallelized, because of the fact that automatic parallelization does not work when algorithms in loops are complicated including subroutine calls. etc. As a consequence, the speedup resulting from the automatic parallelization is unrecognizable. For conventional BIEM algorithms, however, automatic parallelization has been found to be useful, giving the speedup that is not much different from the one achieved with OpenMP. See Nishimura *et al*[8] for more details.

4.3 Parallelization with OpenMP

We next try a parallelization with OpenMP. As we shall see the OpenMP parallelization is performed successfully with minimum changes in the structure of the FMM.

In the upward pass, the loop structure is changed from a children based one to a parents based one in order to prevent the multipole moments from being overwritten at once by more than one thread. For the preconditioning, the block diagonal preconditioning is used here. No change of structure is needed in the downward pass and in the preconditioning process.

Parallel pseudocodes for the upward pass, downward pass and block diagonal preconditioning may be given as follows:

```
C UPWARD PASS

DO level=maxlevel−1 ,1, −1

!$OMP PARALLEL DO
    DO cell = all the cells of the current level

        DO child_cell = all the children of cell

            if (child_cell is a leaf) then
                evaluate M(child_cell)
            endif

            if (level ≠ 1) then
                add M(child_cell) to M(cell)
                after the shift of the center
                (M2M)
            endif

        ENDDO

    ENDDO
!$OMP END PARALLEL DO

ENDDO


C DOWNWARD PASS

DO level=2, maxlevel

!$OMP PARALLEL DO
    DO icell = all the cells of the current level

        if (level ≠ 2) then
            get L(icell) from L(parent_icell)
            after the shift of the center (L2L)
        endif

        DO jcell = all the cells in the interaction
                   list of icell

            add M(jcell) to L(icell) with M2L
        ENDDO

        DO jcell = all the cells in the neighborhood
                   of icell

            if (either icell or jcell is a leaf) then
                evaluate Ax with the direct
                method
            endif

        ENDDO

        if (icell is a leaf) then
            evaluate Ax from L(icell)
        endif

    ENDDO
!$OMP END PARALLEL DO

ENDDO


C BLOCK DIAGONAL PRECONDITIONING

!$OMP PARALLEL DO
DO j=1, num_of_blocks
    evaluate the influence matrix A^j_block,
    calculate (A^j_block)^{-1} and use it
    as the preconditioner

ENDDO
!$OMP END PARALLEL DO
```

where A^j_{block} denotes the jth block diagonal of the matrix A.

As shown in the pseudocodes, the parallelization is the parent based one in the upward pass and the child based one in the downward pass. The block diagonal preconditioning is parallelized for each block. In this parallel algorithm, different threads will never overwrite the same memory for M, L or A^j_{block}, and because the memory accesses by different threads are kept well-separated, little false sharing will occur.

5. Numerical Examples

In this section we test the performance of the parallel FMM codes by solving two dimensional Laplace crack problems and three dimensional elastostatic inclusion problems. An HPC2500 of ACCMS of Kyoto University which has 96 CPUs (1.3GHz) and 384GB of memory is used for the crack problem. The numerical examples of the inclusion analysis are obtained with another HPC2500 of ACCMS with 128 CPUs (1.56GHz) and 512 GB of memory. Double precision arithmetics is used in all the numerical examples shown here, and GMRES (generalized minimal residual method) with no restarts is used for solving the nonsymmetric and dense linear system of equations.

The tolerance in the solution for GMRES is set equal to 10^{-5}.

5.1 Crack Problem for Laplace's Equation in 2-D

Many cracks are distributed uniformly in the infinite two dimensional space as shown in Fig. 4.

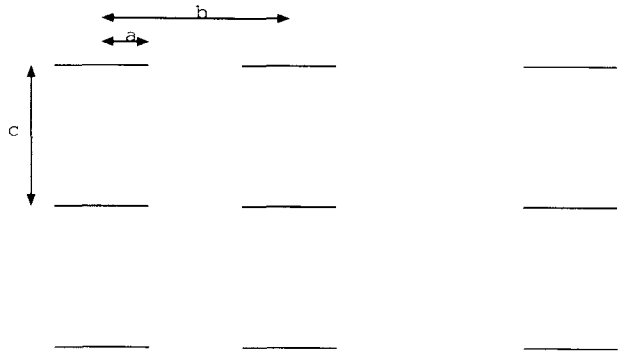

Fig. 4 Crack distribution.

We consider a 70×70 array of cracks, whose geometric size in Fig.4 is set as a:b:c = 2:5:5. Each crack is discretized into 200 constant elements, with graded meshes near its edges; Therefore the number of unknowns is $70 \times 70 \times 200 = 980.000$. All the integrals are evaluated analytically. The infinite series in the multipole and local expansions are truncated at 10 terms. The maximum number of elements in each leaf is set equal to 100. The block diagonal preconditioning corresponding to leaves is applied from the right (See Nishida and Hayami [9] for related attempts). Convergence is achieved after 18 iterations. For the crack problem, GMRES is parallelized with OpenMP. The total elapsed time vs the number of threads is shown in Fig. 5 and the The speedup with the number of threads is shown in Fig.6.

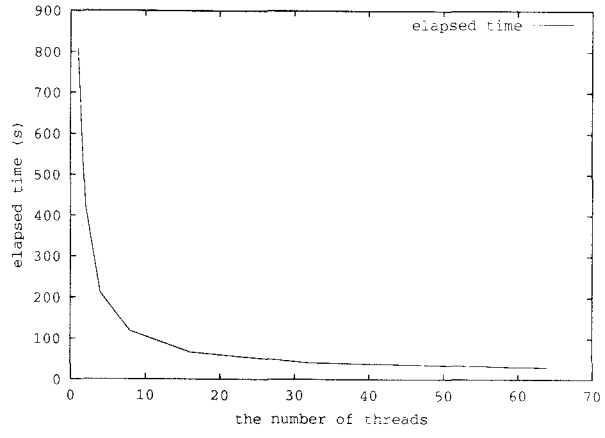

Fig. 5 Elapsed time(s) vs number of threads.

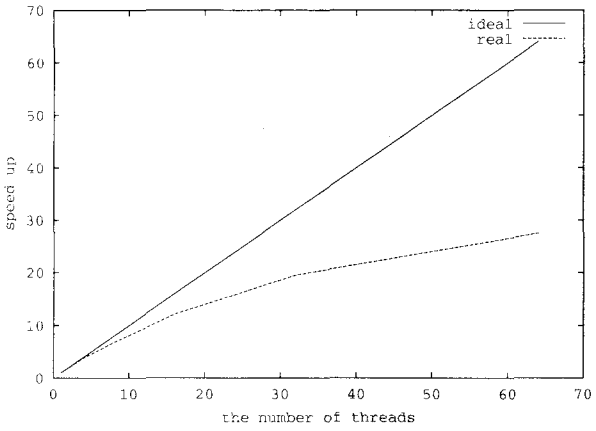

Fig. 6 speedup vs number of threads.

5.2 Elastostatic Inclusion Problem in 3-D

An analysis of an elastic full space with embedded rigid inclusions is carried out. This analysis is intended as a model of carbon nanotube (CNT) based composite materials, taking into consideration the fact that the stiffness of CNT is usually more than an order of magnitude higher than that of the matrix. This problem is formulated as a three dimensional elastostatic bounary value problem with a Dirichlet type boundary condition including unknown rigid-body displacements given as follows:

$$\nu_{0i}^I + (\omega^I \times x)_i = \sum_J \int_{S_J} \Gamma_j^{(i)}(x-y) t_j(y) dS_y + u_i^\infty \quad \text{on} \quad S^I \quad (16)$$

$$\int_{S_I} t_i^I dS = 0 \quad \text{on} \quad S^I \quad (17)$$

$$\int_{S_I} (x \times t^I)_i dS = 0 \quad \text{on} \quad S^I \quad (18)$$

where S denotes the surface of the inclusions, $\Gamma_j^{(i)}$ denotes the fundamental solution of elastostatics in 3-D, t_i is the traction, u_i^∞ is the solution when no inclusions are considered and ν_{0i}^I and ω_i^I are constants. For simplicity, all the inclusions are considered to be the same in shape, the ratio of the height and the diameter of which are set to 5:1. In case 1 (case 2) 32 (128) inclusions are embedded along the x_1 direction in an infinite elastic medium as shown in Fig.7. For the far-field stress, a constant stress σ_{x_1} is applied ($\sigma_{x_2} = \sigma_{x_3} = 0$). Each inclusion is discretized to 456 piecewise constant plane triangle elements; therefore the total DOF is $3 \times 456 \times 32 = 43776$ in case 1 and $3 \times 456 \times 128 = 175104$ in case 2. Also, the integrals in the multipole moments are evaluated with the Gaussian integration and the integrals in the direct calculation are evaluated analytically. The infinite series which appear in the multipole and local expansions are truncated at 20 terms. The maximum number of

Table 2 Elapsed time vs number of threads in inclusion analysis in case 1(32 inclusions)

No. threads	1	2	4	8	16	32	64
D(s)(per iteration)	221.80	112.44	57.271	29.692	15.610	8.7048	5.456
U(s)(per iteration)	6.8012	3.5375	1.7672	0.9857	0.5259	0.3396	0.2666
P(s)(per iteration)	1.0749	0.6792	0.2792	0.1986	0.1826	0.1720	0.1679
Total time (s)	2070	1054	539	283	153	89	60.4

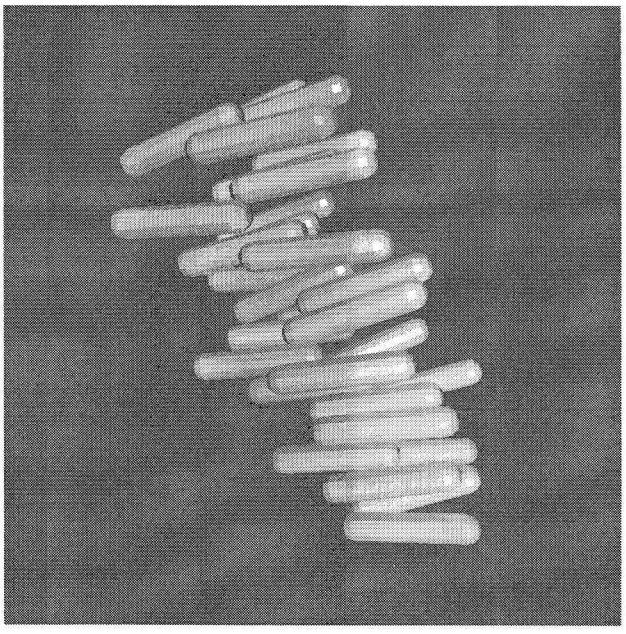

Fig. 7 The embbeded carbon nanotubes in the elastic medium.

elements in the leaf is set equal to 100. For the preconditioning we use the right preconditioning with the block diagonal corresponding to the influence matrix of the inclusion. The preconditioning process is parallelized for each inclusion with OpenMP. GMRES is parallelized with the automatic parallelization.

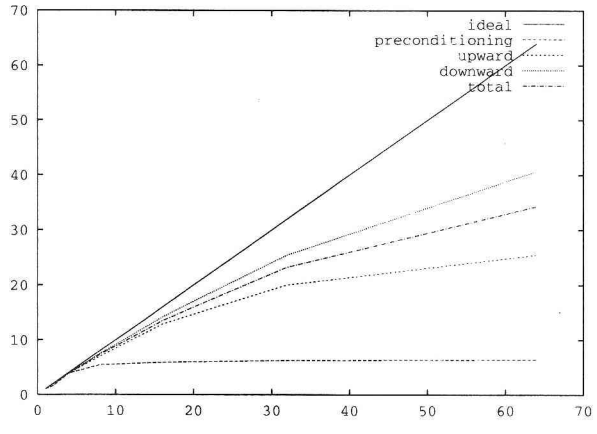

Fig. 8 speedup vs number of threads in case 1.

Convergence is achieved after 7 iterations for both

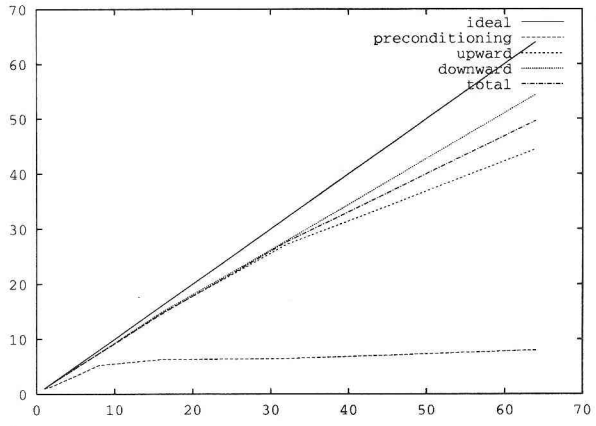

Fig. 9 speedup vs number of threads in case 2.

case 1 and case 2. First, from Table 2 and Table 3 we can say that the elapsed times basically follow the $O(N)$ estimate of the computational complexity of the fast multipole method.

As shown in Table 2 and Table 3 the elapsed time needed for solving the problem is always reduced as the number of threads is increased, but because of the nature of loop based parallelization, the increase of the speedup with the number of threads slows down from around 32 threads. From Fig.8 and Fig.9 it can be seen that for larger systems or for more expensive parts (downward process here) the effect of parallelization becomes even more pronounced. Having reached as high as 50 in speedup with 64 CPUs, we can say that the efficiency of this parallel algorithm is satisfactory considering Amdahl's law given by:

$$\text{speedup} = \frac{1}{(1-P) + P/N} \tag{19}$$

where N is the number of threads and P is the ratio of the parallel program in time. Further, by comparing Fig.6 with Fig.9, we can say that the speedup with the number of threads is more remarkable in the 3-D elastostatic problem than in 2-D problems. This difference occurs because of the nature of the tree structure; namely, the oct-tree structure is more suitable for parallelization than the quad-tree because the more the number of branches is the larger the grain size becomes.

Table 3 Elapsed time vs number of threads in inclusion analysis in case 2(128 inclusions)

No. threads	1	2	4	8	16	32	64
D(s)(per iteration)	745.69	381.52	196.03	100.19	50.163	25.805	13.714
U(s)(per iteration)	23.52	12.00	6.098	3.170	1.601	0.867	0.529
P(s)(per iteration)	4.230	3.016	1.593	0.809	0.673	0.656	0.529
Total time (s)	6962	3573	1840	944.5	478.2	252.1	140.3

6. Conclusion

Through the development, implementation and numerical experiments of parallel fast multipole method, the following conclusions are obtained: the fast multipole accelerated boundary integral equation method can be parallelized on SMP computers. without extraordinary efforts. In spite of the relatively small amount of efforts for programmers and the small change in the program structure, the performance achieved here is not inferior to that obtained with the distributed memory machines [10]. The excellence of the parallel method lies not only in the ease of the parallelization but also in the fact that this parallel fast multipole algorithm is more powerful for larger problems or in three dimensional problems. A limitation of this parallelization method in solving much larger problems is that the number of CPUs cannot exceed the number of CPUs in one node (128 in ACCMS). We shall therefore investigate hybrid parallelization with MPI and OpenMP to use more than one node so that one can solve much larger problems, having, say, hundreds of millions of unknowns.

7. Acknowledgment

The authors would like to express their gratitude to Prof. Yijun Liu of Department of Mechanical, Nuclear and Industrial Engineering of University of Cincinnati for directing our attention to CNT researches.

REFERENCES

1) N. Nishimura: Fast multipole accelerated boundary integral equation methods, *Appl. Mech. Rev.*, Vol.55, pp.299-324, 2002.
2) V.Rokhlin: Rapid solution of integral equations of classical potential theory, *J.Comp. Phys., 60*, pp.187–207, 1985.
3) L. Greengard: *The Rapid Evaluation of Potential Fields in Particle System*, The MIT Press, Cambridge, MA., 1987.
4) http://www.openmp.org/
5) http://www-unix.mcs.anl.gov/mpi/
6) S. W. Song and R. E. Baddour: Parallel processing for boundary element computations on distributed systems, *Engineering Analysis with Boundary Elements*, Vol.19, pp.73–84, 1997.
7) N. Inoue, K. Yoshida, N. Nishimura and S. Kobayashi: A parallel implementation of fast multipole boundary integral equation method, *Proc. Conf. Comp.Eng. Sci. JSCES*, Vol.6-1, pp.211–214, 2001 (in Japanese).
8) N. Nishimura and Y. Otani: On the parallelisation of BIEM for shared memory computers, *Journal of Boundary Element Methods, JASCOM*, Vol.20, pp.83–86, 2003 (in Japanese).
9) T. Nishida and K. Hayami: Application of the panel clustering method to the three-dimensional elastostatic problem, *Boundary Elements XIX (Eds. M.Marchetti et al.)*, pp.613–622, Comp. Mech. Publ., Southhampton, 1997.
10) K. Yoshida and N. Nishimura: On fast method computing potentials, *Trans ACCMS*, Kyoto Univ, Vol.2, pp.123–128, 2003 (in Japanese).

(Received April 16, 2004)