A Parallel Multilevel Fast Multipole Algorithm Based On OpenMP

Wei-Chao Pi, Xiao-Min Pan, and Xin-Qing Sheng[1]
Center for Electromagnetic Simulation, School of Information and Electronics,
Beijing Institute of Technology, 100081, P. R. China.
(Email: peao_kelvin@163.com; xmpan@bit.edu.cn; xsheng@bit.edu.cn)

***Abstract*-A parallel multilevel fast multipole algorithm (MLFMA) based on OpenMP is proposed for shared memory parallel platform. According to requirements of efficient OpenMP parallelization and different numerical characteristics of different parts in MLFMA, the parallelization scheme is carefully designed on the establishment of near-field interaction matrix and on the far-field interaction in matrix vector multiplication in MLFMA. Numerical experiments show that the parallel MLFMA is efficient and has a consistent accuracy with the sequential MLFMA.**

I. INTRODUCTION

Parallel technique is a subject of increasing interest to improve computational capability with the fast development of computer science. High efficient parallel algorithm is a hot issue in computational electromagnetics (CEM) community because associated applications are always resources (including CPU-time and memory) demanding. A series of successes have actually been reached in this area. Parallelization of the multilevel fast multipole algorithm (MLFMA) [1, 2] is the most distinguished one among them. The fast multipole method (FMM) and its recursive variant, the MLFMA have been considered as the breakthrough of the CEM, which significantly advance the capability of electromagnetic simulations for large-scale problems [3]. With the MLFMA, problems with the scale of about tens of wavelengths can be solved. Combined with parallel techniques, a set of parallel schemes on the MLFMA have been proposed. The capability of MLFMA has been furtherly improved to be several thousands of wavelengths.

These works on parallel MLFMA are all based on message passing interface (MPI) for distributed memory parallel platform [1, 2]. Programs based on MPI are highly scalable and portable. But because the workload distribution and communications should be carefully designed by researchers, developments of high efficient parallel algorithm based on MPI are considerably challenging. Lots of skills and experiences are required to reach acceptable efficiency, especially for the complicated algorithm such as the MLFMA. In contrast, OpenMP standard provides us a simple substitutive [4]. OpenMP often works on share memory computer and has a different parallel mechanism from that of MPI. Parallelization in OpenMP is carried out on threads, and is done on processes in MPI. Time-consuming communications are thus avoided in OpenMP since different threads can share a same block of memory.

With maturity and popularity of the multi-core technique, shared memory system are the main trend on modern personal computers and server clusters. How to take advantage of this is a valuable issue. Some works have been reported to employ OpenMP to improve the parallel efficiency of finite difference time domain (FDTD) and finite element method (FEM) [5,6]. But little has been done on the MLFMA. In this paper, a parallel MLFMA is proposed based on OpenMP. According to requirements of efficient OpenMP parallelization and different numerical characteristics of different parts in the MLFMA, a parallelization scheme is carefully designed on the establishment of near-field interaction matrix and on the far-field interaction in matrix vector multiplication in the MLFMA.

II. OPENMP PARALLELIZATION

OpenMP is an application program interface (API) that may be used to explicitly direct ***multi-threaded, shared memory parallelism*** [4]**.** The multi-thread property distinguishes OpenMP from MPI, where parallelization is implemented on process, as shown in fig.1.

A process is the unit of an executing application. It is started by the operating system when an application is launched. The process owns the memory, resources, and threads of execution that are associated with a running instance of an executable program. When the process is started, one thread is initially associated with that process. As long as one thread continues to be associated with the process, the process continues to run.

[1] This work was supported by the NSFC under Grant 10832002, by the NSFC under Grant 60901005, by the excellent young scholars Research Fund of Beijing Institute of Technology under Grant 2008Y0102.

Every process contain at least one thread, is called primary thread.

A thread is the smallest schedulable unit of execution in an application. A thread is always associated with a particular process—after the thread is started, it never runs in the context of another process. Although many simple applications use only a single thread, it's not uncommon for more complex applications to use multiple threads over the lifetime of the process. In fact, OpenMP uses multiple threads to realize parallelism.

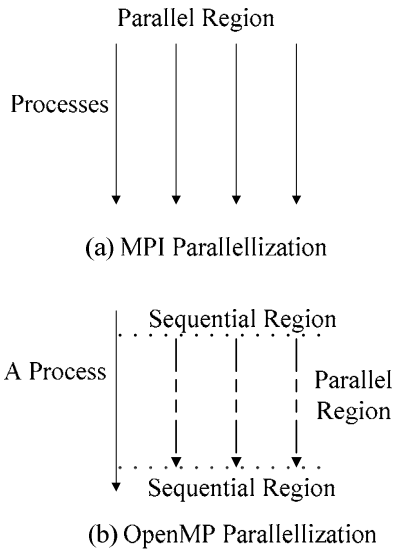


Fig.1 Difference between MPI and OpenMP parallelization

In OpenMP parallel programming, the OpenMP specification defines a set of pragmas. A pragma is compiler directives on how to process the block of code that follows the pragma. The most basic pragma is the **#pragma omp parallel** to denote a ***parallel region***. Since parallelization can be implemented by a set of pragma directives, OpenMP really provides us a portable, scalable model with a simple and flexible interface for developing parallel applications on platforms from the desktop to the supercomputer. However for complex algorithms, OpenMP parallelization should be carefully designed to obtain good efficiency. As pointed out by Quinn [4], if a loop does not have enough iterations, the time spent forking and joining threads may exceed the time saved by dividing the loop iterations among multiple threads. Among many considerations for OpenMP performance, two important ones should be emphasized.

The first one is the data protection. In OpenMP, all threads share the same memory space. This means that all threads access the same memory. The data is not security when some threads try to write it concurrently. Data protection is therefore required in OpenMP programming, which is always realized by two manners. On one hand, parallelization should be implemented on appropriate regions. This is especially important for algorithm with complex structure, such as the MLFMA. In the following section III, we will discuss this in detail. On the anther hand, data protection can be realized by declaring the scope of the associated variables as ***private***. If a variable is scoped as ***private***, each thread has its own copy of the variable.

```
!$OMP PARALLEL
      ....
      !$OMP DO
      ....
      !$OMP END DO
      ....
!$OMP END PARALLEL

!$OMP PARALLEL
      ....
      !$OMP DO
      ....
      !$OMP END DO
....
!$OMP END PARALLEL
```

(a) the first realization

```
!$OMP PARALLEL
      ....
      !$OMP DO
      ....
      !$OMP END DO
      .....
      !$OMP DO
      ....
      !$OMP END DO
!$OMP END PARALLEL
```

(b) the second realization

Fig.2 Different realizations of OpenMP parallelization

Another important consideration is to parallelize at the highest level possible, such as outer DO/FOR loops. That's to say, enclose multiple loops in one parallel region. In general, parallel regions should be made as large as possible to reduce parallelization overhead. As shown in fig.2, the former parallelization is less efficient than the latter.

Due to the limited space, we refer the reader to more considerations for efficient OpenMP parallelization in the literature [4].

III. MLFMA and Its OpenMP Parallelization

A. Multilevel Fast Multipole Algrithm

For perfectly-conducting objects, discretizations of surface integral equations lead to $N \times N$ (where N is the number of unknowns) dense matrix equations in the form of

$$y = Zx, \tag{1}$$

where the matrix elements for Z can be interpreted as electromagnetic interactions of discretization elements, i.e., basis and testing functions. The matrix equation (1) can be solved iteratively via a Krylov subspace algorithm, where the required matrix-vector multiplication (MVM) can be accelerated by the fast multipole algorithm (FMM) or MLFMA

[3]. The FMM/MLFMA decomposes MVM into two parts: near-field interaction (NFI) and far-field interaction (FFI). The former is computed directly, while the latter is accelerated by FMM/MLFMA [3]. Thus the matrix equation in the context of FMM/MLFMA has a form of

$$y = Z_N x + Z_F x, \tag{2}$$

where Z_N and Z_F are, respectively, near part and far part of Z. The entries in matrix Z_N remain the same as the method of moment, but those corresponding to Z_F are not numerically available. In fact, FFI $Z_F x$ in the FMM is realized through 3 steps: the aggregation, the translation and the disaggregation. In the MLFMA, interpolation/anterpolation combined with center-shifting operations is required to transfer far-field patterns from sub-box to parent-box or vice versa.

To present a whole picture of the implementation of MLFMA, we employ a multilevel presentation from coarser to finer considerations. Fig. 3 shows a three-level development procedure for the sequential implementation of the MLFMA. The following will discuss how to efficient parallelize the MLFMA on OpenMP platforms.

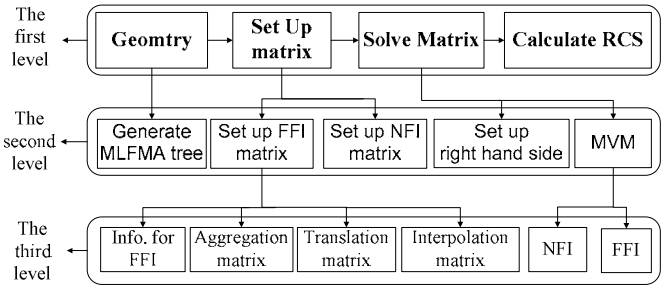


Fig. 3 Multilevel development of the MLFMA

Table I CPU-time used at different phases in the MLFMA

	Proportion
Establishment of matrixes in the MLFMA	
Info. of Geometry	0.52%
FFI matrix	5.49%
NFI matrix	88.37%
RCS calculation	6.07%
One MVM in the MLFMA	
NFI	6.36%
FFI	93.63%

B. OpenMP Parallelization

As mentioned in section II, a straightforward parallelization on the MLFMA may lead to inefficiency. A careful analysis on numerical characteristics of different parts in MLFMA therefore is very beneficial. To this end, a numerical experiment is performed on a perfect electrical conducting (PEC) sphere with a diameter of 20 wavelengths (λ), denoted by *sph-20*. *Sph-20* involves 295,788 unknowns with an average mesh size of 0.12 λ. The resource used in the computation is listed in table I. Apparently, in matrix establishment phase, filling NFI matrix consumes most of the CPU-time. In contrast, FFI takes up about 94% time of that in one MVM. So our parallel scheme will constrict to parallelization of the NFI matrix establishment and FFI implementation in MVM.

Fig.4 Parallelization for NFI matrix establishment

```
!$OMP PARALLEL

    !$OMP DO i=1, N; for each edge
        DO j=1, N; for each edge
            Calculate the matrix element Z_ij
        ENDDO
    !$OMP END DO

!$OMP END PARALLEL
```

Fig.5 Parallelization for FFI

```
DO i= levmax-1, 2

    !$OMP PARALLEL
    !$OMP DO j=1, M; for each box in i-th level
        Aggregation operations
    !$OMP END DO

    !$OMP END PARALLEL
ENDO

DO i= 2, levmax

    !$OMP PARALLEL
    !$OMP DO j=1, M; for each box in i-th level
        Translation operations
    !$OMP END DO
    !$OMP END PARALLEL

    !$OMP PARALLEL
    !$OMP DO j=1, M; for each box in i-th level
        Disaggregation operations
    !$OMP END DO
    !$OMP END PARALLEL

ENDO
```

For the establishment of near-field interaction matrix, parallelization is done on the top DO loop according to the consideration discussed in section II, as shown in Fig.4.

For the implementation of FFI in MVM, more considerations should be taken into. Fig.5 depicts our parallel scheme, where "levmax" denotes finest level used in the MLFMA. On one hand, the parallelization can't be done at the top level DO loop because "levmax" is usually not larger than 20. On the other hand, the translation and disaggregation phases must be parallelized in consequence because a memory recycling strategy is used to store the far-field pattern [1].

IV. NUMERICAL EXPERIMENTS

Computation on the *sph-20* is used to validate our proposed OpenMP parallel strategy for the MLFMA. The experiments are carried out on the *Deep-Comp 7000* HPC at the Chinese Academy of Sciences [7]. RWG functions are chosen as basis and testing functions to discretize CFIE with a combination

coefficient of 0.5. The GMRES iteration process is terminated when the 2-norm of the residual vector is reduced to 10^{-3}.

Table III. CPU-time used when different number of threads employed

No. of threads	1	2	4	8	16
NFI-matrix(s)	217.19	108.12	54.18	27.94	14.20
MVM(s)	12.17	5.70	3.13	1.70	0.97
Total	17m18s	9m54s	6m43s	4m57s	4m6s

In the calculations, we change the number of threads. Table III lists time used and Fig.6 presents the radar cross section (RCS) results from parallel MLFMA against that from the sequential MLFMA. Since results obtained from parallel MLFMA are almost identical, only the result for the case with 2 threads is plotted. As shown in fig.6, there is no loss of accuracy in our OpenMP parallelization. Fig.7 shows the acceleration rates for the NFI matrix establishment and MVM implementation(results is obtained under static schedule). The matrix establishment is well parallelized with an acceleration rate of 16 even when 16 threads are used. On the other hand, high acceleration rate of MVM is also observed in Fig.7.

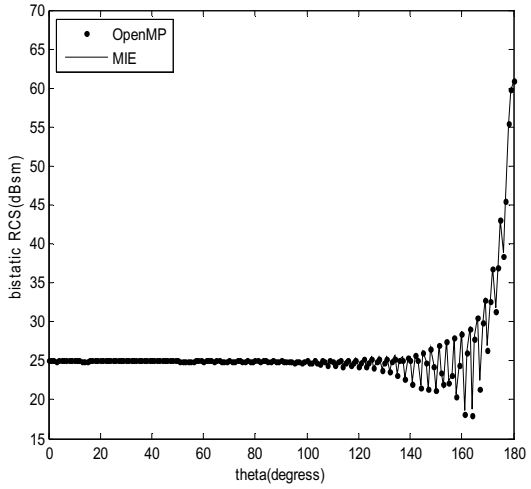
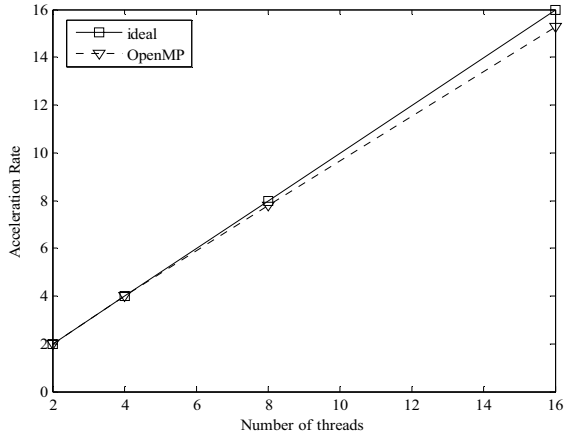
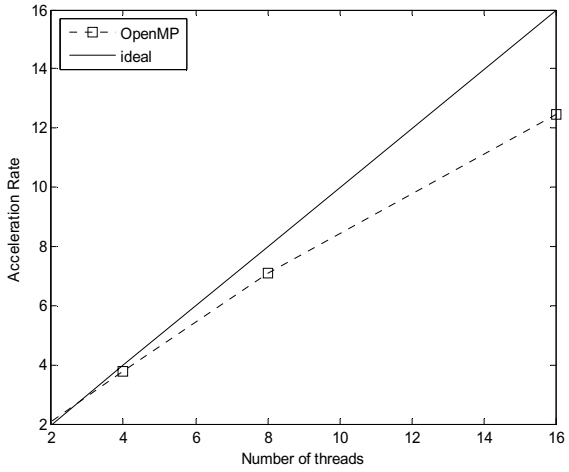


Fig.6 RCS for *sph-20*



(a) Establishment of NFI matrix



(b) MVM

Fig. 7 Parallel acceleration rate of OpenMP

V. CONCLUSIONS

A parallel multilevel fast multipole algorithm (MLFMA) based on OpenMP is proposed and implemented for shared memory parallel platform. According to requirements of efficient OpenMP parallelization and the different numerical characteristics of different parts in MLFMA, the parallelization scheme is carefully designed on the establishment of near-field interaction matrix and on the far-field interaction in matrix vector multiplication in the MLFMA. Numerical experiments show that the parallel MLFMA is efficient and has a consistent accuracy with the sequential MLFMA.

REFERENCES

[1] X. M. Pan, and X. Q. Sheng, "A sophisticated parallel MLFMA for scattering by extremely large targets," *IEEE Antennas Propag. Mag.*, vol. 50, no. 3, pp. 129-138, June, 2008.

[2] Ö. Ergul, Gurel, L., "A hierarchical partitioning strategy for an efficient parallelization of the multilevel fast multipole algorithm," IEEE Trans. Antennas Propag., vol. 57, no. 6, pp. 1740 - 1750, June, 2009.

[3] J. M. Song, C. C. Lu, and W. C. Chew, "MLFMA for electromagnetic scattering by large complex objects," IEEE Trans. Antennas Propagat., vol. 45, pp. 1488-1493, Oct. 1997.

[4] M. J. Qinn, Parallel programming in C with MPI and OpenMP, McGraw Hill, 2004

[5] Y. Liu, Z. Liang, and Z. Yang, "A novel FDTD approach featuring two-level parallelization on pc cluster," *Progress In Electromagnetics Research*, vol.80, no. pp.393-408, 2008.

[6] G. A. Gravvanis, "OpenMP based parallel normalized direct methods for sparse finite element linear systems", *The Journal of Supercomputing*, vol.47, no.1, pp. 44–52, Jan. 2009

[7] http://www.sccas.cn/gb/index.html