Progress In Electromagnetics Research, Vol. 112, 199–213, 2011

ON OPENMP PARALLELIZATION OF THE MULTILEVEL FAST MULTIPOLE ALGORITHM

X.-M. Pan, W.-C. Pi, and X.-Q. Sheng

Center for Electromagnetic Simulation
School of Information and Electronics
Beijing Institute of Technology, 100081, China

Abstract—Compared with MPI, OpenMP provides us an easy way to parallelize the multilevel fast multipole algorithm (MLFMA) on shared-memory systems. However, the implementation of OpenMP parallelization has many pitfalls because different parts of MLFMA have distinct numerical characteristics due to its complicated algorithm structure. These pitfalls often cause very low efficiency, especially when many threads are employed. Through an in-depth investigation on these pitfalls with analysis and numerical experiments, we propose an efficient OpenMP parallel MLFMA. Two strategies are proposed in the parallelization, including: 1) loop reorganization for far-field interaction in the MLFMA; 2) determination of a transition level. Numerical experiments on large scale targets show the proposed OpenMP parallel scheme can perform as efficiently as the MPI counterpart, and much more efficiently than the straightforward OpenMP parallel one.

1. INTRODUCTION

Combining integral equation based methods [1–10] with parallel techniques becomes a new trend in computational electromagnetic (CEM) community because associated applications are always computational resources (including both CPU time and memory) demanding. A series of successes have actually been reached in this area [11–26]. Parallelization of the fast multipole method (FMM) and its recursive variant, the multilevel fast multipole algorithm (MLFMA) [12–22] is the most distinguished one among them. To date, most works on the parallelization of the FMM/MLFMA are purely

Received 8 December 2010, Accepted 1 January 2011, Scheduled 13 January 2011
Corresponding author: Xiao-Min Pan (xmpan@bit.edu.cn).

based on message passing interface (MPI) for distributed memory parallel platforms [12–20]. MPI programs are highly scalable and portable. But its implementation is challenging because workload distribution and communications should be manually realized. Lots of skills and experience are required to guarantee acceptable efficiency. In contrast, OpenMP standard provides us an easy alternative [27]. OpenMP often works on shared-memory computer systems and has a totally different parallel mechanism from MPI. The time-consuming communications in MPI can be avoided because OpenMP realizes parallelism through multithreading.

Some works have employed OpenMP to parallelize the FMM and its variants [21, 22], exhibiting the potential of OpenMP. In [21], OpenMP is combined with MPI to accelerate the FFT extension of the conventional FMM (denoted by FMM-FFT). Although the proposed parallel FMM-FFT has a relatively high parallel efficiency, the OpenMP parallel strategy in [21] can hardly be applied to the MLFMA because of the distinct algorithm structures between the MLFMA and the FMM-FFT. At the same time, a straightforward implementation of the OpenMP FMM may perform awkwardly. As shown in [22], the parallel efficiency is only about 40% when 8 threads are used, and decreases to be less than 25% when the number of threads increases to 16. Due to the distinct algorithm structures between the MLFMA and the FMM (FMM-FFT), pitfalls and essentials on OpenMP parallelization of the MLFMA are not discussed in [21, 22]. On this observation, OpenMP parallel MLFMA are studied in detail. Two strategies are proposed to guarantee high efficiency of OpenMP parallel MLFMA:

1) Strategy of loop reorganization
   Our study shows that the conventional implementation of loops computing far-field interaction in MLFMA results in extremely low parallel efficiency of OpenMP parallel MLFMA. A strategy for reorganization of loops is proposed to improve the parallel efficiency of OpenMP parallel MLFMA.
2) Strategy of determining transition level
   It is important to find a transition level in the proposed OpenMP scheme when different parallel strategies are employed in the different levels in MLFMA. A strategy on determining the transition level is proposed to obtain high efficiency of OpenMP parallel MLFMA.

An OpenMP MLFMA is developed and numerical experiments are carried out to validate our analysis. The capability of the parallel MLFMA is demonstrated by calculating RCS from two extremely large targets.

2. OUTLINE OF THE MLFMA

For perfectly electric conducting (PEC) objects, discretization of surface integral equations yields $N \times N$ (where N is the number of unknowns) dense matrix equations in the form of

$$[Z]\{J\} = \{E^i\}, \tag{1}$$

where $[Z]$ is the impedance matrix, $\{J\}$ is the effective currents, and $\{E^i\}$ is the incident wave. The matrix Equation (1) can be solved iteratively, where the required matrix-vector multiplication (MVM) can be accelerated by the FMM or MLFMA [1]. 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. Thus the matrix equation in the context of FMM/MLFMA has a form of

$$[Z]_N\{I\} + [Z]_F\{I\} = \{E^i\}, \tag{2}$$

where $[Z]_N$ and $[Z]_F$ are, respectively, the near part and the far part of $[Z]$. The entries in matrix $[Z]_N$ remain the same as the method of moments, but those corresponding to $[Z]_F$ are not numerically available. In fact, FFI $[Z]_F\{I\}$ in the FMM is realized through three stages: the aggregation, the translation and the disaggregation. In the MLFMA, interpolation/anterpolation combined with center-shifting operations is required to transfer far-field patterns (FFPs) from son-box to parent-box or vice versa.

3. PITFALLS IN THE OPENMP PARALLEL MLFMA AND OUR SOLUTIONS

It is valuable to maintain the good virtue of being simple by figuring out suitable computational regions for OpenMP parallelization. To this end, numerical experiments are performed on a perfect electrical conducting (PEC) sphere with a 48 wavelengths (λ) diameter, denoted by *sph-48*. *Sph-48* involves 2,639,532 unknowns with an average mesh size of 0.10λ. In all computations, a 9-level MLFMA (the 0-th level is the coarsest level) is employed. Table 1 lists proportions of CPU-time used for different stages in the MLFMA. In the matrix filling stage, filling NFI matrix consumes most of CPU-time. In contrast, FFI takes up about 98% of time for one MVM. So parallelization will concentrate on these two stages. It should be noted that the translation stage takes up about 60% of CPU-time for one FFI, while aggregation and disaggregation stages cost the rest 40%. Although how much time a stage in the MLFMA consumes exactly may be implementation

Table 1. Proportions of CPU-time used for different stages in the MLFMA.

	Proportion
Establishment of matrixes in the MLFMA	
FFI matrix	2.43%
NFI matrix	95.94%
Other operations	$< 1.7\%$
one MVM in the MLFMA	
NFI	2.09%
FFI	97.91%

dependent, the NFI matrix filling and FFI implementation stages are sure to be the most time-consuming ones. For both these two stages, OpenMP parallelization is carried out on the top DO loop to reach high parallel efficiency.

A straightforward OpenMP MLFMA performs reasonably well for moderately sized targets. With the size of the target increasing, the efficiency decreases rapidly when more threads are employed. The reason lies in that the number of boxes at coarse levels may be less than that of threads. Inspired by the hybrid MPI parallel scheme [13–18], a strategy is developed in this paper, where a transition level is chosen and different parallel methods are employed at coarse (levels with larger boxes than those at the transition level) levels and fine ones. In specific, parallelization is carried out on the loops over FFPs at coarse levels, and on the loops over boxes at other levels.

3.1. The Loop Reorganization Strategy

Pitfalls may arise if the OpenMP parallel directives is directly moved from loops over boxes to those over FFPs. Loops are required to be reorganized for high efficiency. For translation stage, it requires few modifications on the reorganization, but for aggregation/disaggregation careful considerations are necessary.

Aggregation/disaggregation involves boxes at two levels: A parent level and the son level. Because parent boxes often need more FFPs than son boxes, interpolation/anterpolation is required to transfer FFPs between parent and son levels. Therefore, aggregation/disaggregation requires three loops: one over parent boxes, one over son boxes and one over FFPs. All these loops can serve as an outer loop in OpenMP implementation. Unfortunately,

an unsuitable choice of an outer loop would definitely result in *race condition* in OpenMP and thus a bad efficiency. This can be illustrated by the following examples as shown in Fig. 1.

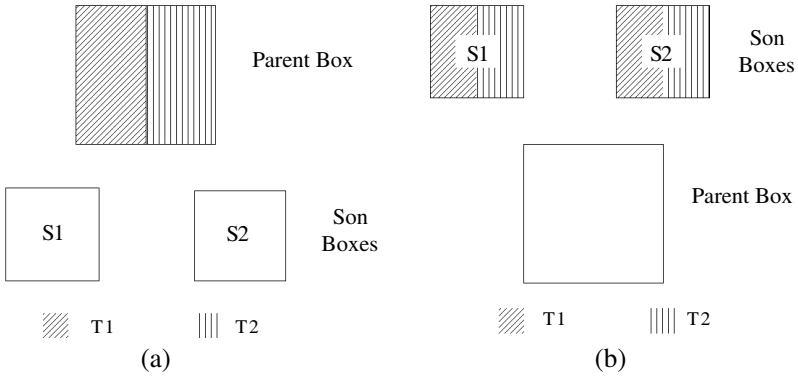

Figure 1. Parallel strategy for disaggregation. (a) Unsuitable strategy. (b) Suitable strategy.

Suppose the loop on parent boxes is treated as an outer loop and two threads are involved in. The parent box has two sons, as shown in Fig. 1. There are three facts for the case of Fig. 1(a) during the computation. First, FFPs (or to say loop cycles) of the parent box are assigned to two threads (T1 and T2) equally. Second, FFPs of the son boxes, denoted by S1 and S2, are shared by T1 and T2. Third, both parts of FFPs assigned to T1 and T2 contribute to FFPs of the son boxes S1 and S2. As a result, T1 and T2 may access S1 or S2 to update the corresponding FFP values simultaneously. Unfortunately, this sets up a race condition, in which the computation exhibits nondeterministic behavior. To avoid the race condition, a *critical section* can be employed to protect the update process. But it will cause a poor speedup because the critical section is actually a piece of sequential code. Another solution is to protect the update process by claiming variables for S1 and S2 as *reduction* ones. Thus, OpenMP will take care of the details, such as storing FFPs for S1 and S2 in private variables and then adding partial sums to the shared variables after the loop. However, additional memory and floating operations are required in this solution. Consequently, the speeding up rate is also limited, especially when a large number of threads are used. Our smart strategy is to set the loop over son boxes as the outer loop, as shown in Fig. 1(b). The main idea is to partition FFPs of son boxes, which will be updated during the computation, to different threads. All threads need not access and write these FFPs of son boxes that are

```
C...Aggregation
DO i=levmax-1, 2
   !$OMP PARALLEL
   !$OMP DO k=1, M; each FFP of a parent box
      Do j=1, P; each box at the i-th level
         Do n=1, S; each son box of the k-th box
            Computations on aggregation
         END DO
      END DO
   !$OMP END PARALLEL
ENDO
```

```
C...Translation and disaggregation
DO i= 2, levmax
   !$OMP PARALLEL
   !$OMP DO j=1, M; each FFP for box in the i-th level
      DO k=1, Q; each box at the i-th level
         The loop over boxes for translation
      END DO
      IF i>2 THEN
         DO k=1, Q; each box at the i-th level
            Computations on disaggregation
         END DO
      END IF
   !$OMP END DO
   !$OMP END PARALLEL
END DO
```

Figure 2. New parallel scheme of FFI at coarse levels.

not allocated to them. Therefore, race condition is totally avoided. On the contrary, FFPs of parent boxes should be allocated among threads in the aggregation stage for the similar reason. To implement the above strategy, a careful organization of computational steps on loops should be designed as shown in Fig. 2.

3.2. The Transition Level Determination Strategy

The choice of a transition level is vital for efficiency and two factors should be taken into account. First, CPU time used in the sequential code should be minimized since changes on the computational loops will exhibit different efficiencies. Fig. 3 presents the time used for computations at different levels in one FFI when the outer loop is, respectively, the loop over boxes and the loop over FFPs, denoted by *Loop-Box* and *Loop-FFP*. One thread is employed in these computations. It indicates that efficiencies of these two cases are comparable at coarse levels, but the *Loop-Box* case is much more efficient than the *Loop-FFP* one at fine levels. It is thus not suitable to choose a fine level as a transition level. According to this, a criterion for the choice of a transition level l_{trans} can be written as

$$l_{trans} < l_{\max} - q, \tag{3}$$

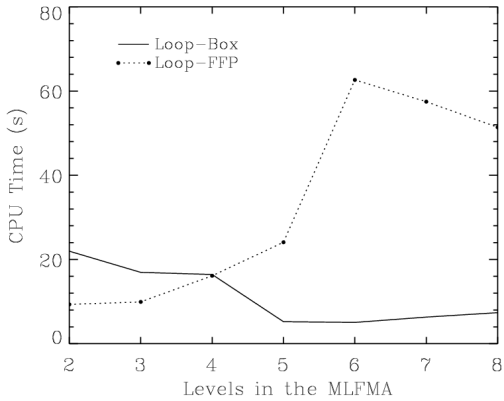

Figure 3. CPU time used for the computations at different levels in one FFI.

where $l_{\max}$ refers to the finest level in the MLFMA (the level with only one box is indexed by 0), q is an integer greater than or equal to 1. Since computational speed depends heavily on hardware architecture, q may vary even for a same problem on different computers. It is set to be 3 according to our numerical experiments on moderately sized and large scale targets.

Second, a transition level should be chosen by maximizing parallel efficiency. It is therefore required that boxes at coarse levels should have enough FFPs as the loop over FFPs is the outer loop. Based on this, a criterion for the choice of a transition level can be written as:

$$S_{l_{trans}}/T \geq w, \quad \text{and} \quad \mathrm{S}_{l_{trans}+1}/T < w, \tag{4}$$

where $S_{l_{trans}}$ denotes the number of boxes at the l_{trans} level, T is the number of threads, and w is the number of cycles (FFPs) that should be allocated to each thread for good efficiency. How many loop cycles are enough for each thread may vary in different applications. As far as the MLFMA is concerned, at least 20 cycles (FFPs) are needed by one thread according to our numerical experiments. So, we usually set w to be 20. (3) and (4) may leads to different choices, and the coarser one is usually chosen as the transition level in our implementation. In practical, choice obtained from (4) is generally the finally adopted one since the proposed scheme is always used for large scale targets where a large number of threads are employed. From this point of view, choice of q is generally not a key factor in the proposed scheme.

4. NUMERICAL EXPERIMENTS

Numerical experiments are carried out to study the proposed OpenMP parallel MLFMA. All the experiments are carried out on the high performance computing (HPC) platform *Deep-Comp 7000* at the Chinese Academy of Sciences [28]. The computational node used to conduct our numerical experiments is configured with 8 Xeon quad-core CPUs and 256 GB memory. 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}.

Let t_1 be the time for evaluating a single FFI on a single thread and let t_T be the time for computing a FFI on T threads. Then, the parallel efficiency is defined as

$$\eta = \frac{t_1}{T \times t_T} \times 100\% \tag{5}$$

Note that in an ideal situation $\eta = 100\%$. Because the parallel efficiency of NFI matrix filling can always reach as high as over 95%, in the following discussion, we focus on the parallelization of FFI.

The *sph-48* is used to test the OpenMP parallel strategy discussed in Section 3. And then two large scale targets: a sphere with a diameter of 220λ and a complex airplane model with a largest dimension of 530λ, are used to demonstrate the capability of the OpenMP parallel MLFMA.

4.1. Investigation on the Schedule Manners

Different from MPI, OpenMP provides us a mechanism of automatic workload distribution. It is this attractive virtue that makes OpenMP programming much simpler than MPI. There are three types of schedule manners in the automatic workload distribution mechanism. Namely, they are: *static, dynamic* and *guided schedules* [27]. The effect of different schedule manners on parallel efficiency is investigated by changing the number of threads. Since translation takes up 60% CPU time of FFI, we firstly investigate parallel efficiency of translation at different levels. In our computations, different schedule manners are employed by fixing the number of threads to be 4. As shown in Fig. 4, different schedule manners show distinct efficiency while the guided one performs much better than the other two. This difference is actually caused by the distinct numerical characteristics at different levels in the MLFMA. The amount of work associated with a given box may always differ much from that with another box at a same level. A static schedule cannot take it into account because loop cycles (boxes)

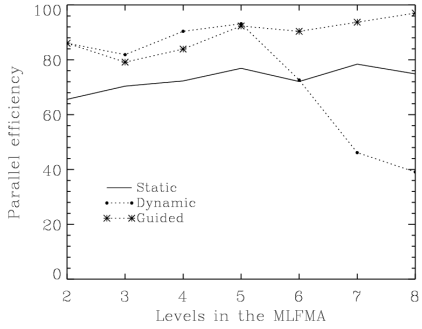

Figure 4. Parallel efficiency of translation at different levels when 4 threads are used.

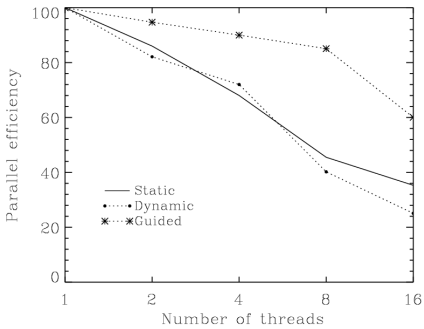

Figure 5. Parallel efficiency as a function of the number of threads.

are distributed among threads equally before the computation begins. Load imbalance thus arises. Both dynamic and guided schedules can avoid the load imbalance since they assign workload dynamically. But a dynamic schedule needs more scheduling operations than a guided one when a same chunk-size is used. In addition, the more cycles a loop has, the more overhead will be in a dynamic schedule. This explains why a dynamic schedule performs inferior to a guided one in our numerical experiments, as shown in Fig. 4. In particular, a dynamic schedule degrades the efficiency seriously at fine levels with a large number of boxes/cycles. It is worthy to point out that specifying an integer other than 1 (the default value) to the chunk-size for guided schedule doesn't improve the total efficiency greatly. On one hand, a small chunk-size will not be adopted to distribute workload since OpenMP would find an optimal chunk-size automatically [27]. On the other hand, a large chunk-size may lead to load imbalance since workload of boxes may differ much even for boxes at a same level.

In Fig. 5, we present the parallel efficiency on FFI as a function of the number of threads. Again, it can be found that a guided schedule performs much better than the other two.

4.2. Validation on the Loop Reorganization and Transition Level Determination Strategies

As discussed in Section 3, a straightforward parallel scheme can only perform well when a small number of threads are employed. In fact, it can be observed from Fig. 5 that the efficiency decreases rapidly with the increasing of threads. New strategies should be used to overcome this deficiency.

Here, the strategy on reorganization of loops is firstly validated. Since the reason for inverting loops is clear, we only need validate the strategy to choose an outer loop for aggregation/disaggregation. Table 2 compares the CPU time consumed by disaggregation before and after our strategy is used. It can be seen that a suitable organization of loops by our proposed strategy can reach high efficiency while the unsuitable one performs awkwardly. In fact, the computation cannot be completed in reasonable time if more than 4 threads are used because of race condition. Here, the transition level is fixed at the 3rd level.

The criteria to choose a transition level, described in (3) and (4), are also investigated. Three cases are studied in the experiments where a transition level is determined by different manners: 1) case 1: by the strategy proposed in Section 3.2; 2) case 2: by manually fixing it to be the 2nd level; 3) case 3: by manually fixing it to be the 5th level. Fig. 6 presents the parallel efficiency as a function of the number of threads. As shown in Fig. 6, our strategy can reach an overall good performance. In case 3, efficiency is extremely low compared with the

Table 2. CPU time (seconds) consumed by disaggregation before and after our strategy is used (* means that the computation cannot be completed in reasonable time).

Threads	Before	After
1	15.9	13.4
2	30.1	7.0
4	*	3.7

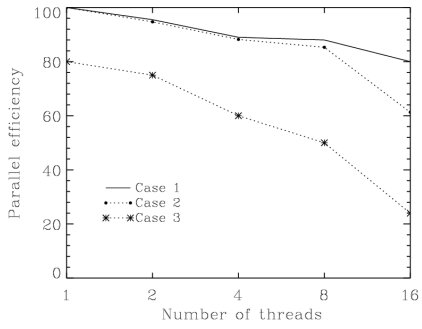

Figure 6. Parallel efficiency as a function of the number of threads.

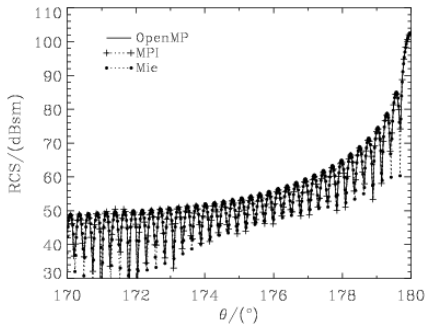

Figure 7. Bistatic RCS from *sph-220*.

other two cases due to the unsuitable choice of a transitional level. As shown in Fig. 3, the CPU time may increase a lot even if the program runs sequentially in this case.

4.3. Capability of the Proposed Parallel Scheme

A PEC sphere with a diameter of 220λ is employed to demonstrate the capability of our proposed OpenMP parallel MLFMA. This sphere, denoted by *sph-220*, involves 4,4930,700 unknowns with an average mesh size of 0.10λ. In the computation, an 11-level MLFMA, where the 0-th level is the coarsest level, is employed with a requirement of about 180 GB memories. In order to compare our OpenMP parallelization with the MPI one, we also calculate radar cross section (RCS) from this sphere by the MPI code [16]. The MPI code requires 230 GB memory when 32 processes are employed since each process needs some additional memory space. This is one of the largest spheres which can be solved on our computational platform. Fig. 7 presents the RCS results from OpenMP parallel MLFMA against that from the MPI parallel one. Since RCS results obtained from all the computations are almost identical, only those for the 32 threads (processes in terms of MPI) cases are plotted. The Mie theory result is also presented in Fig. 7. As shown in Fig. 7, there is no loss of accuracy in our OpenMP parallelization.

Figure 8 illustrates the parallel efficiency as a function of the number of threads/processes. The efficiency of the MPI parallel MLFMA is also presented in the figure for comparison. As can be found from the figure, our proposed OpenMP MLFMA exhibits a high efficiency and performs as efficiently as the MPI counterpart.

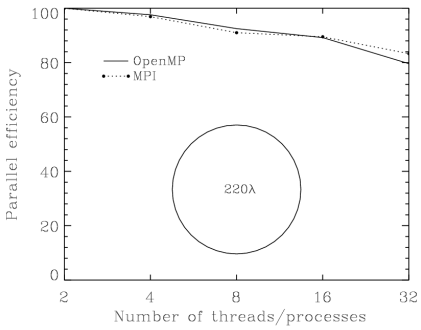

Figure 8. Parallel efficiency as a function of the number of threads.

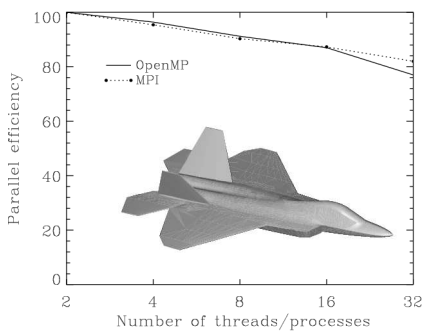

Figure 9. Parallel efficiency as a function of the number of threads.

The total CPU time used is about 5 hours and 4 hours, respectively, in the 32 threads case and in the 32 processes case. The OpenMP MLFMA requires more time because only two major parts of MLFMA are parallelized.

To further demonstrate the capability of our OpenMP MLFMA, RCS from an airplane model is calculated. It is a complex target and has a largest dimension of 530λ, involving 46,443,538 unknowns with an average mesh size of 0.13λ. A 12-level MLFMA is employed with the usage of over 200 GB memories. Fig. 9 presents the OpenMP parallel efficiency against the MPI one. Again, the efficiency of OpenMP is very high, which is comparable to that of MPI. The total CPU time used is about 4 hours and 3.5 hours, respectively, in the 32 threads case and in the 32 processes case. The OpenMP MLFMA requires more time for the same reason as in the *sph-220* case.

From above numerical experiments, it can be found that the straightforward OpenMP parallel scheme can only work well for moderately sized targets when no more than 8 threads are used, while the one with carefully designed outer loops can perform well even when 32 threads are used.

At the same time, it is worthy to point out that OpenMP and MPI differ much in mechanisms, which have their advantages and disadvantages. For example, OpenMP parallelization is simple but it is limited by hardware resources, while a MPI one has good scalability yet is complex. The best way of parallelization may combine them together, especially for HPC clusters with computational nodes of shared-memory systems.

5. CONCLUSIONS

Although OpenMP provides us an easy way to parallelize the MLFMA on shared-memory systems, the implementation of OpenMP parallelization has many pitfalls because different parts of the MLFMA have distinct numerical characteristics due to its complicated algorithm structure. The strategies on loop reorganization and on determination of transition level are proposed to guarantee high efficiency of OpenMP parallel MLFMA. The proposed OpenMP scheme is efficient while maintaining the good virtue of being simple. Numerical experiments on large scale targets show that the proposed OpenMP MLFMA can perform as efficiently as the MPI counterpart, and much more efficiently than the straightforward OpenMP scheme. Because of the current restriction on hardware resources of shared-memory systems, the capability of the OpenMP parallel MLFMA is still limited. A combination of OpenMP and MPI should be a best choice for extremely

large scale targets by fully exploiting advantages of both shared-memory systems and distributed memory ones.

ACKNOWLEDGMENT

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, by the Basic Research Found of Beijing Institute of Technology under Grant 20090542001.

REFERENCES

1. Song, J. M., C. C. Lu, and W. C. Chew, "MLFMA for electromagnetic scattering by large complex objects," *IEEE Trans. Antennas Propagat.*, Vol. 45, 1488–1493, Oct. 1997.
2. Ayestaran, R. G., J. Laviada-Martinez, and F. Las-Heras, "Realistic antenna array synthesis in complex environments using a Mom-SVR approach," *Journal of Electromagnetic Waves and Applications*, Vol. 23, No. 1, 97–108, 2009.
3. Lai, B., N. Wang, H. B. Yuan, and C. H. Liang, "Hybrid method of higher-order MoM and nyström disretization PO for 3D PEC problems," *Progress In Electromagnetics Research*, Vol. 109, 381–398, 2010.
4. Hou, Z. G., C. Wang, and H. C. Yin, "Multilevel thresholding method for a sparse representation of reduced matrix in CBFM," *Journal of Electromagnetic Waves and Applications*, Vol. 24, No. 17–18, 2605–2614, 2010.
5. Ling, J., S. X. Gong, S. T. Qin, W. T. Wang, and Y. J. Zhang, "Wide-band analysis of on-platform antenna using MoM-PO combined with maehly approximation," *Journal of Electromagnetic Waves and Applications*, Vol. 24, No. 4, 475–484, 2010.
6. Ergul, O. and L. Gurel, "Improving iterative solutions of the electric-field integral equation via transformations into normal equations," *Journal of Electromagnetic Waves and Applications*, Vol. 24, 2129–2138, 2010.
7. Ping, X. W., T. J. Cui, and W. B. Lu, "The combination of bcgstab with multifrontal algorithm to solve febi-mlfma linear systems arising from inhomogeneous electromagnetic scattering problems," *Progress In Electromagnetics Research*, Vol. 93, 91–105, 2009.

8. Peng, Z., X. Q. Sheng, and F. Yin, "An efficient twofold iterative algorithm of Fe-Bi-MLFMA using multilevel inverse-based ilu preconditioning," *Progress In Electromagnetics Research*, Vol. 93, 369–384, 2009.
9. Eibert, T. F., Ismatullah, E. Kaliyaperumal, and C. H. Schmidt, "Inverse equivalent surface current method with hierarchical higher order basis functions, full probe correction and multi-level fast multipole acceleration," *Progress In Electromagnetics Research*, Vol. 106, 377–394, 2010.
10. Wu, G., X. Zhang, and B. Liu, "A hybrid method for predicting the shielding effectiveness of rectangular metallic enclosures with thickness apertures," *Journal of Electromagnetic Waves and Applications*, Vol. 24, No. 8–9, 1157–1169, 2010.
11. Cui, Z., Y. Han, Q. Xu, and M. Li, "Parallel MoM solution of jmcfie for scattering by 3-d electrically large dielectric objects," *Progress In Electromagnetics Research M*, Vol. 12, 217–228, 2010.
12. Donepudi, K. C., J. M. Jin, S. Velamparambil, J. M. Song, and W. C. Chew, "A higher order parallelized multilevel fast multipole algorithm for 3-D scattering," *IEEE Trans. Antennas Propag.*, Vol. 49, 1069–1078, Jul. 2001.
13. Velamparambil, S., W. C. Chew, and J. M. Song, "10 million unknowns: Is it that big?" *IEEE Antennas Propagat. Mag.*, Vol. 45, 43–58, Apr. 2003.
14. Velamparambil, S. and W. C. Chew, "Analysis and performance of a distributed memory multilevel fast multipole algorithm," *IEEE Trans. Antennas Propagat.*, Vol. 53, 2719–2727, Aug. 2005.
15. Pan, X. M. and X. Q. Sheng, "A highly efficient parallel approach of multi-level fast multipole algorithm," *Journal of Electromagnetic Waves and Applications*, Vol. 20, No. 8, 1081–1092, 2006.
16. Pan, X. M., and X. Q. Sheng, "A sophisticated parallel MLFMA for scattering by extremely large targets," *IEEE Antennas Propag. Mag.*, Vol. 50, No. 3, 129–138, Jun. 2008.
17. Ergül, Ö. and L. Gürel, "Efficient parallelization of the multilevel fast multipole algorithm for the solution of large-scale scattering problems," *IEEE Trans. Antennas Propagat.*, Vol. 56, 2335–2345, Aug. 2008.
18. Ergül, Ö. and L. Gürel, "A hierarchical partitioning strategy for an efficient parallelization of the multilevel fast multipole algorithm," *IEEE Trans. Antennas Propag.*, Vol. 57, No. 6, 1740–1750, Jun. 2009.

19. Li, W. D., W. Hong, and H.-X. Zhou, "An IE-ODDM-MLFMA scheme with DILU preconditioner for analysis of electromagnetic scattering from large complex objects," *IEEE Trans. Antennas Propag.*, Vol. 56, 1368–1380, May 2008.
20. Yang, M. L. and X. Q. Sheng, "Parallel high-order Fe-Bi-MLFMA for scattering by large and deep coated cavities loaded with obstacles," *Journal of Electromagnetic Waves and Applications*, Vol. 23, No. 13, 1813–1823, 2009.
21. Taboada, J. M., M. G. Araujo, F. Obelleiro, J. M. Bertolo, L. Landesa, J. Rivero, and J. L. Rodriguez, "Supercomputer aware approach for the solution of challenging electromagnetic problems," *Progress In Electromagnetics Research*, Vol. 101, 241–256, 2010.
22. Buchau, A., S. M. Tsafak, W. Hafla, W. M. Rucker, "Parallelization of a fast multipole boundary element method with cluster OpenMP," *IEEE Trans. Magn.*, Vol. 44, 1338–1341, Jun. 2008.
23. Zhang, H. W., X. W. Zhao, Y. Zhang, D. G. Donoro, W. X. Zhao, and C. H. Liang, "Analysis of a large scale narrow-wall slotted waveguide array by parallel MoM out-of-core solver using the higher order basis functions," *Journal of Electromagnetic Waves and Applications*, Vol. 24, No. 14–15, 1953–1965, 2010.
24. Gao, P. C., Y.-B. Tao, and H. Lin, "Fast RCS prediction using multiresolution shooting and bouncing ray method on the GPU," *Progress In Electromagnetics Research*, Vol. 107, 187–202, 2010.
25. Jiang, W. Q., M. Zhang, and Y. Wang, "CUDA-based radiative transfer method with application to the em scattering from a two-layer canopy model," *Journal of Electromagnetic Waves and Applications*, Vol. 24, No. 17–18, 2509–2521, 2010.
26. Tay, W. C., D. Y. Heh, and E. L. Tan, "Gpu-accelerated fundamental adi-FDTD with complex frequency shifted convolutional perfectly matched layer," *Progress In Electromagnetics Research M*, Vol. 14, 177–192, 2010.
27. Qinn, M. J., *Parallel Programming in C with MPI and OpenMP*, McGraw Hill, 2004.
28. http://www.sccas.cn/gb/index.html.