Parallel Implementation of the Steepest Descent Fast Multipole Method (SDFMM) on a Beowulf Cluster for Subsurface Sensing Applications

D. Jiang, W. Meleis, M. El-Shenawee, E. Mizan, M. Ashouei, and C. Rappaport

***Abstract*—We present the parallel, MPI-based implementation of the SDFMM computer code using a thirty two-node Intel Pentium-based Beowulf cluster. The SDFMM is a fast algorithm that is a hybridization of the method of moments (MoMs), the fast multipole method (FMM), and the steepest descent integration path (SDP), which is used to solve large-scale linear systems of equations produced in electromagnetic scattering problems. An overall speedup of 7.2 has been achieved on the 32-processor Beowulf cluster and a significant reduced runtime is achieved on the 4-processor 667 MHz Alpha workstation.**

***Index Terms*—Fast algorithms, parallelization, SDFMM, subsurface sensing.**

I. Introduction

THE SDFMM was originally developed at the University of Illinois at Urbana Champaign to analyze large-scale three dimension (3-D) scattering problems [1]–[3]. Recently, its computer code has been successfully modified to handle subsurface sensing applications, in particular, the scattering from a PEC and/or penetrable spheroid buried under a two dimensional randomly rough ground surface [4], [5]. The SDFMM has computational complexity for CPU time and computer memory equal to only $O(N)$ per iteration versus $O(N^2)$ for the method of moments (MoMs), where N is the total number of the unknowns [1]. The significantly reduced complexity of the SDFMM over several other computational electromagnetics techniques has enabled efficient Monte Carlo simulation studies [5]. Additional speedup is desirable for increased Monte Carlo sample size or for inverse scattering applications. In this work, we used the MPI library for the parallel implementation of the SDFMM code [6]–[8].

II. Parallelization

The SDFMM is used to solve the linear system of equations given by [1]–[5]:

$$\overline{\overline{Z}}\,\overline{I} = \overline{V} \tag{1a}$$

where $\overline{\overline{Z}}$ is the impedance matrix, $\overline{I}$ is the vector of unknown coefficients of the electric and magnetic surface currents, and $\overline{V}$ is associated with the incident waves on the rough ground surface. The matrix $\overline{\overline{Z}}$, which is filled in the MoM formulations, becomes sparse with using the SDFMM and the system of equations in (1a) can be written as

$$\overline{\overline{Z}}'\overline{I} + \overline{\overline{Z}}''\overline{I} = \overline{V}. \tag{1b}$$

The sparse matrix $\overline{\overline{Z}}'$ has its nonzero elements calculated and stored using the conventional MoM, which are then multiplied by the vector $\overline{I}$ (near field interactions) while the matrix–vector multiply $\overline{\overline{Z}}''\overline{I}$ is computed in one step without calculating or storing any elements of the matrix $\overline{\overline{Z}}''$. This is achieved by using the fast multipole method (FMM) hybridized with the steepest descent integration path (SDP).

The following three bottlenecks in the SDFMM code can benefit from being parallelized: i) the subroutines that calculate the elements of the sparse matrix $\overline{\overline{Z}}'$; ii) the subroutines that execute the matrix–vector multiplication $\overline{\overline{Z}}'\overline{I}$ in each iteration of the solver; iii) the subroutines that execute the fast multipole method for $\overline{\overline{Z}}''\overline{I}$ (far field interactions).

The computer code has been parallelized by exploiting the underlying available data parallelism. The key data structure in subroutine i) is the sparse matrix $\overline{\overline{Z}}'$, which is stored as blocks of nonzero elements. These blocks are distributed among all processors, and no additional communication is needed. When this routine is parallelized, we achieved near-linear speedups on 32 processors. In the matrix–vector multiplication $\overline{\overline{Z}}'\overline{I}$, the computation is parallelized by distributing $\overline{I}$ to all processors in each iteration. The resulting vector components produced by the multiplication are then distributed to all processors. For bottleneck iii), there are two involved subroutines to compute the far field interactions consisting of a series of loops with complex interdependences. Each loop is separately parallelized, with collective communication used to distribute the results to all processors after executing each subroutine. In addition, these two subroutines are executed in parallel, followed by subsequent distribution of the results to all processors. Load balance between these two subroutines is achieved using a detailed performance model based on the serial execution time of each routine, the time required for collective communication operations, and the amount of communication overhead needed. The structure of the parallelized SDFMM application is shown in Fig. 1.

We evaluated the parallel implementation of the SDFMM computer code on a 32-node Intel Pentium-based Beowulf

Manuscript received July 18, 2001; revised October 21, 2001. This work was supported by the ERC Program of the NSF under Award EEC-9986821, in part by the ARO Demining MURI Grant DAA 0-55-97-0013, and by the College of Engineering at the University of Arkansas. The review of this letter was arranged by Associate Editor Dr. Arvind Sharma.

D. Jiang, W. Meleis, E. Mizan, M. Ashouei, and C. Rappaport are with the Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 USA (e-mail: meleis@ece.neu.edu).

M. El-Shenawee is with the Department of Electrical Engineering, University of Arkansas, Fayetteville, AR 72701 USA (e-mail: magda@uark.edu).

Publisher Item Identifier S 1531-1309(02)00869-3.

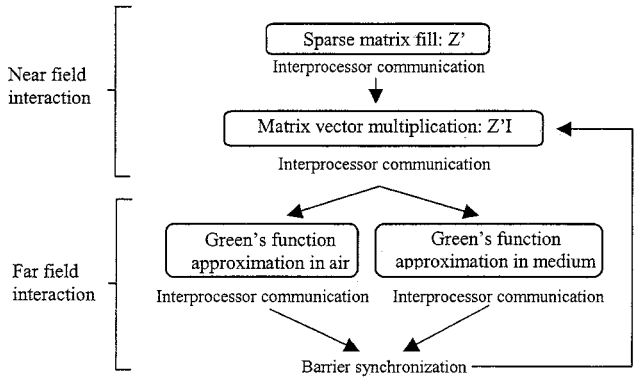

Fig. 1. Structure of the parallelized SDFMM application.

cluster. Thirty one nodes of the Beowulf cluster are 350 MHz Intel Pentium IIs with 256 MB of RAM and one node is a 4×450 MHz Intel Pentium II Xeon shared memory processor with 2 GB of RAM. The nodes are connected to a 100 BaseTX Ethernet network and they use the SuSE 6.1 operating system with Linux kernel 2.2.13, and the MPICH 1.2.1 implementation of the MPI library. We also tested the parallelized code on a 4-node shared memory Compaq Alpha-based workstation (667 Mhz Alpha 21 264) of 16 GB total RAM. The processor uses the UNIX OSF/1 V5.1 operating system with the MPICH 1.1.2 MPI library.

Our benchmark includes three small-scale cases executed on the 256 MB Intel cluster, and in addition one moderate-scale case that is executed on the Alpha workstation. All results obtained by executing the parallel version of the code are validated with those computed by the serial version of the code [4], [5]. The scattering problem configurations used in [5] are employed here, but for only one rough surface realization. The rough ground (characterized by Gaussian statistics with zero mean for the height), is described by the rms height (σ) and the correlation length (l_c). In all cases, the relative dielectric constant of the ground soil (dry sand) and the penetrable buried object (TNT in a land mine) are $\varepsilon_r = 2.5 - j0.18$ and $\varepsilon_r = 2.9 - j0.0092$, respectively, and the ground correlation length is $l_c = 0.5\lambda_0$. A Gaussian beam with horizontal polarization is employed for the incident waves [5]. In Case 2, the buried sphere has radius of $a = 0.16\lambda_0$ with burial depth equal to $z = -0.32\lambda_0$ measured from its center to the mean plane of the ground, while in Case 3 and 4 the buried spheroid has dimensions $a = 0.3\lambda_0$ and $b = 0.15\lambda_0$, and is buried at $z = -0.3\lambda_0$. The ground dimensions are $3\lambda_0 \times 3\lambda_0$ in Cases 1–3 and $8\lambda_0 \times 8\lambda_0$ in Case 4. Table I summarizes the parameters and output results for Cases 1–4.

The speedup of a parallelized application is defined as the ratio of the serial runtime to the parallel runtime. In Fig. 2, the overall speedup and the speedup for the initialization routine (filling matrix $\overline{\overline{Z}}'$) are plotted versus the number of processors for Case 1. The speedup curves for Cases 2 and 3 are similar, with slightly different peak values of 6.2 and 7.2, respectively. The results show the significant speedup in the initialization time that is needed to fill the sparse matrix $\overline{\overline{Z}}'$. This initialization speedup affects the overall speedup of the code. In each case, the peak overall speedup is observed when running on 32 processors, but most of this speedup is achieved using only 12 processors.

TABLE I
OVERALL SPEEDUP FOR CASES 1-4

Case #	# of Unknowns	σ	Object	System	# of Processors	Serial/Par. Time (min.)	Speedup (overall)
1	8,800	$0.3\lambda_o$	None	Cluster	32	99/14	7.1
2	8,800	$0.1\lambda_o$	Sphere	Cluster	32	90/14	6.2
3	8,800	$0.04\lambda_o$	Spheroid	Cluster	32	88/12	7.2
4	60,320	$0.04\lambda_o$	Spheroid	Alpha Server	4	96/37	2.5

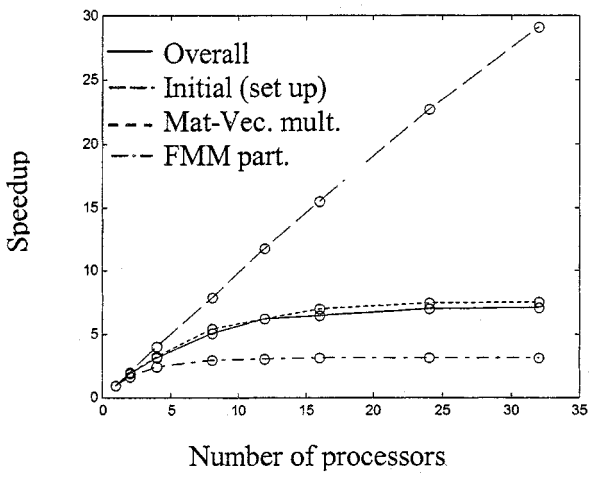

Fig. 2. Speedup for the three separate bottlenecks in the code, versus the number of processors on the Beowulf cluster.

The efficiency of an application for a given number of processors is defined as the ratio of the speedup to the number of processors. Over Cases 1–3, the average speedup on 32 processors is 6.8, giving an efficiency of 0.21. Based on the serial runtimes, 88% of the code is executed in parallel. Therefore, by Amdahl's Law [9], the peak speedup achievable for the current parallelization of the code is 8.3. We conclude that communication overhead and load imbalance among the processors accounts for the reduction in speedup from 8.3 to 6.8.

A comparison between the speedups achieved in the other bottlenecks i)–iii) is also shown in Fig. 2. These results demonstrate that the overall speedup is almost the same as that achieved in the matrix–vector multiplication $\overline{\overline{Z}}'\overline{I}$ which is the bottleneck in ii).

In the second set of experiments, we solved the moderate-scale problem of Case 4 (60 320 unknowns) on the Alpha SMP using all four available processors. The overall speedup in this case is 2.5, which is close to the predicted peak speedup of 2.9. This implies that executing the parallel code on the 4-Alpha 667 MHz processor gives an impressive reduced absolute runtime for this moderate-scale case. The serial version of the code requires 950 MB of memory, while the parallel version requires 1154 MB of memory distributed over four processors (288, 290, 289 and 287 MB each). The results of the parallel solution were identical to those of the serial implementation presented in [5].

The results described in this section demonstrate that by exploiting fine-grained parallelism within a single surface realiza-

tion (one run of the code), we have achieved speedups. However, when the number of rough surface realizations is much larger than the number of available processors, as with Monte Carlo simulations, larger speedups are possible. This situation occurs when we need to run Monte Carlo simulations [5]. In this case, we assign a group of these realizations (runs of the code) to be executed in parallel on each processor. Since the computations are independent and little communication is needed, this coarse-grained parallelism gives a perfect speedup that is only limited by the number of available processors. In other subsurface scattering configurations, we may need to obtain multiple views of a target buried under the same rough surface realization [4], which requires running the code only few times. A combination of fine and coarse-grained parallelism can make efficient use of all available processors.

III. Conclusions

MPI is successfully employed for the parallel implementation of the SDFMM. A significant overall speedup of 7.2 has been achieved on the 32-processor Beowulf cluster and a dramatic reduced runtime is gained using the 4-processor Alpha workstation. The greatest potential for speedup occurs in the sparse matrix filling step.

Acknowledgment

The authors would like to thank Prof. W. Chew and Prof. E. Michielssen at UIUC for allowing them the use and modification of the SDFMM computer code for the current application.

References

[1] V. Jandhyala, "Fast multilevel algorithms for the efficient electromagnetic analysis of quasiplanar structures," Ph.D. dissertation, Dept. Elect. Comput. Eng., Univ. Illinois, Urbana-Champaign, 1998.

[2] V. Jandhyala, B. Shanker, E. Michielssen, and W. C. Chew, "A fast algorithm for the analysis of scattering by dielectric rough surfaces," *J. Opt. Soc. Amer. A*, vol. 15, no. 7, pp. 1877–1885, July 1998.

[3] M. El-Shenawee, V. Jandhyala, E. Michielssen, and W. C. Chew, "The steepest descent fast multipole method (SDFMM) for solving combined field integral equation pertinent to rough surface scattering," in *Proc. IEEE APS/URSI '99 Conf.*, Orlando, FL, July 1999, pp. 534–537.

[4] M. El-Shenawee, C. Rappaport, E. Miller, and M. Silevitch, "3-D subsurface analysis of electromagnetic scattering from penetrable/PEC objects buried under rough surfaces: Use of the steepest descent fast multipole method (SDFMM)," *IEEE Trans. Geosci. Remote Sensing*, vol. 39, pp. 1174–1182, June 2001.

[5] M. El-Shenawee, C. Rappaport, and M. Silevitch, "Monte Carlo simulations of electromagnetic wave scattering from random rough surface with 3-D penetrable buried object: Mine detection application using the SDFMM," *J. Opt. Soc. Amer. A*, Aug. 2001.

[6] "Message passing interface forum. MPI: A message-passing interface-standard," *Int. J. Supercomput. Applicat. High Perform. Comput.*, no. 8, 1994.

[7] S. V. Velamparambil, J. E. Schutt-Aine, J. G. Nickel, J. M. Song, and W. C. Chew, "Solving large scale electromagnetic problems using a linux cluster and parallel MLFMA," in *IEEE Antennas Propagat. Symp.*, vol. 1, July 1999, pp. 636–639.

[8] S. Li, C. H. Chan, L. Tsang, Q. Li, and L. Zhou, "Parallel implementation of the sparse matrix/canonical grid method for the analysis of two-dimensional random rough surfaces (three-dimensional scattering problem) on a Beowulf system," *IEEE Trans. Geosci. Remote Sensing*, vol. 38, pp. 1600–1608, July 2000.

[9] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Spring Joint Comput. Conf. 30*, Atlantic City, NJ, Apr. 1967, pp. 483–485.