CUDA-MPI Implementation of Fast Multipole Method on GPU Clusters for Dielectric Objects

Nghia Tran, Tuan Phan, and Ozlem Kilic

Department of Electrical Engineering and Computer Science
The Catholic University of America, Washington, DC, 20064, USA
16tran@cua.edu, 30phan@cua.edu, kilic@cua.edu

Abstract ─This paper investigates the Fast Multipole Method (FMM) for large-scale electromagnetics scattering problems for dielectric objects. The algorithm is implemented on a Graphical Processing Unit (GPU) cluster using CUDA programming and Message Passing Interface (MPI). Its performance is investigated in terms of accuracy, speedup, and scalability. The details of the implementation and the performance achievements are shown and analyzed, demonstrating a scalable parallelization while maintaining a good degree of accuracy.

Index Terms ─ Dielectric objects, Fast Multipole Method (FMM), Graphics Processing Unit (GPU), Message Passing Interface (MPI).

I. INTRODUCTION

Modelling electrically large dielectric objects plays an important role in the research of target identification or the stealth and anti-stealth technology. The excessively heavy requirements of memory and computational resources of such applications pose a challenging problem in the computational electromagnetics community. The past two decades have witnessed many numerical techniques developed to reduce this burden without significant loss of accuracy, including Adaptive Integral Method (AIM) [1], Impedance Matrix Localization (IML) [2], and Fast Multipole Method (FMM) [3]. Among others, FMM is the most suitable technique for large-scale problems in reducing the computational complexity of the conventional technique, Method of Moment (MoM) [4] from $O(N^3)$ to $O(N^{1.5})$, where N denotes the number of unknowns. Some other approaches such as AIM and IML have the complexity of $O(N^{1.5}logN)$ and $O(N^2logN)$, respectively. Many authors have investigated the parallelization of FMM on CPU clusters for solving the dielectric problems [5]. However, to the best of our knowledge, FMM has not been studied for dielectric electromagnetics problems on GPU clusters. Recently our group has applied single-level FMM for perfect electric conductor (PEC) objects [6]-[7], and good speedup factors compared to the CPU implementations were achieved. However, our previous implementations focused only on PEC objects, which can be limiting for simulating real-life scenarios.

In this paper, we investigate the parallelization of FMM for dielectric electromagnetics structures using the PMCHW formula [8] on a multi-node GPU cluster with CUDA and MPI programing. We demonstrate that the implementation of FMM on GPU clusters outperforms that of the CPU counterpart in terms of speedup and scalability.

The rest of the paper is organized such that Section II provides an overview of FMM for dielectric objects. Section III describes the parallelization of FMM on GPU clusters. Experimental results are discussed in Section IV. Finally, our findings are summarized in the conclusions section.

II. OVERVIEW OF THE FAST MULTIPOLE METHOD FOR DIELECTRIC OBJECTS

In this section, we provide a brief overview to help our discussion on the parallel implementation of FMM for dielectric objects, which is presented in detail in Section III.

FMM solves for the linear equation system created in the form of $ZI = V$, where I represents the unknown currents, V depends on the incident field, and Z is the impedance matrix. The main idea of FMM is the grouping concept as shown in Fig. 1, where N edges in the mesh of a given structure are categorized into M localized groups based on their proximity. According to this approach, two interaction types can be defined: near and far, as depicted in Fig. 1. The Z matrix is split into two components, Z^{near} and Z^{far}, as shown in (1):

$$\sum_{m'}^{M} Z_{mm'} I_{m'} = \sum_{m'}^{M} Z_{mm'}^{near} I_{m'} + \sum_{m'}^{M} Z_{mm'}^{far} I_{m'} = V_m,$$
$$\text{where } Z_{mm'} = \begin{bmatrix} Z_{mm',JJ} & Z_{mm',JM} \\ Z_{mm',MJ} & Z_{mm',MM} \end{bmatrix}, V_m = \begin{bmatrix} E_m \\ H_m \end{bmatrix}, \quad (1)$$

and m and m' are observation and source edges in the mesh, respectively.

The near term comprises of interactions between

Submitted On: July 14, 2016
Accepted On: August 17, 2016

spatially close edges, and is computed and stored in a similar manner to MoM [4]. For dielectric objects, PMCHW formula [8] is used in this paper and the four components of Z^{near} is shown in (2)-(4):

$$Z^{near}_{mm',JJ} = \frac{j\omega\mu}{4\pi}\int_S \boldsymbol{f}_{m'}(\boldsymbol{r}')\int_S G_1(\boldsymbol{r},\boldsymbol{r}')\bullet \boldsymbol{f}_m(\boldsymbol{r})dS'dS + \frac{1}{4\pi\, j\omega\varepsilon}\int_S(\nabla\bullet \boldsymbol{f}_m(\boldsymbol{r}))\int_S G_1(\boldsymbol{r},\boldsymbol{r}')\bullet(\nabla\bullet \boldsymbol{f}_{m'}(\boldsymbol{r}'))dS'dS, \quad (2)$$

$$Z^{near}_{mm',MM} = \frac{j\omega\varepsilon\eta^2}{4\pi}\int_S \boldsymbol{f}_{m'}(\boldsymbol{r}')\int_S G_1(\boldsymbol{r},\boldsymbol{r}')\bullet \boldsymbol{f}_m(\boldsymbol{r})dS'dS + \frac{\eta^2}{4\pi\, j\omega\mu}\int_S(\nabla\bullet \boldsymbol{f}_m(\boldsymbol{r}))\int_S G_1(\boldsymbol{r},\boldsymbol{r}')\bullet(\nabla\bullet \boldsymbol{f}_{m'}(\boldsymbol{r}'))dS'dS, \quad (3)$$

$$Z^{near}_{mm',MJ} = \frac{-\eta}{4\pi}\int_S \boldsymbol{f}_{m'}(\boldsymbol{r}')\int_S \nabla G_1(\boldsymbol{r},\boldsymbol{r}')\times \boldsymbol{f}_m(\boldsymbol{r})dS'dS = -Z^{near}_{mm',JM}. \quad (4)$$

The interactions between the remaining edges that are spatially far from each other constitute the far term. The advantage of separating the Z matrix into two components is that the Z^{far} matrix does not need to be computed and stored ahead of time. Instead it is factorized into the radiation, T^E/T^{ED}, receive, R^E, and translation functions, T_L. Equations (5)-(11) depict these functions based on PMCHW formula:

$$Z^{far}_{mm',JJ} = \frac{\omega\mu k}{16\pi^2}\int d^2\hat{\boldsymbol{k}}\, \boldsymbol{T}^E_{r_{im}}(\hat{\boldsymbol{k}})T_L(k,\hat{\boldsymbol{k}},\boldsymbol{r}_{ii'})\bullet R^E_{m'i'}(\hat{\boldsymbol{k}}), \quad (5)$$

$$Z^{far}_{mm',MM} = \frac{\omega\varepsilon k\eta^2}{16\pi^2}\int d^2\hat{\boldsymbol{k}}\, \boldsymbol{T}^E_{r_{im}}(\hat{\boldsymbol{k}})T_L(k,\hat{\boldsymbol{k}},\boldsymbol{r}_{ii'})\bullet R^E_{m'i'}(\hat{\boldsymbol{k}}), \quad (6)$$

$$Z^{far}_{mm',MJ} = \frac{k\eta}{16\pi^2}\int d^2\hat{\boldsymbol{k}}\, \boldsymbol{T}^{ED}_{r_{im}}{}'(\hat{\boldsymbol{k}})T_L(k,\hat{\boldsymbol{k}},\boldsymbol{r}_{ii'})\bullet R^E_{m'i'}(\hat{\boldsymbol{k}}) = -Z^{far}_{mm',JM}, \quad (7)$$

where

$$T^E_{r_{im}} = \int_S (\boldsymbol{I} - \hat{\boldsymbol{k}}\hat{\boldsymbol{k}})\bullet \boldsymbol{f}_m(\boldsymbol{r}_{im})e^{-j\boldsymbol{k}\bullet\boldsymbol{r}_{im}}dS, \quad (8)$$

$$T^{ED}_{r_{im}} = \int_S \hat{\boldsymbol{k}}\times \boldsymbol{f}_m(\boldsymbol{r}_{im})e^{-j\boldsymbol{k}\bullet\boldsymbol{r}_{im}}dS, \quad (9)$$

$$R^E_{r_{m'i'}} = \int_S \boldsymbol{f}_{m'}(\boldsymbol{r}_{m'i'})e^{-j\boldsymbol{k}\bullet\boldsymbol{r}_{m'i'}}dS, \quad (10)$$

$$T_L = \sum_{l=0}^{L}(-j)^l(2l+1)h_l^{(2)}(\boldsymbol{k}\bullet\boldsymbol{r}_{ii'})P_l(\hat{\boldsymbol{k}}\bullet\boldsymbol{r}_{ii'}). \quad (11)$$

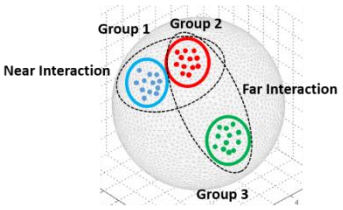


Fig. 1. FMM grouping concept.

In the equations above, the prime syntax denotes the source points, and i refers to the groups in the mesh. The unit vector $\hat{\boldsymbol{k}}$ denotes the K possible field directions in k space, $\boldsymbol{f}(\mathbf{r})$ denotes the basic functions, $h_l^2(r)$ is the spherical Hankel function of the second kind, and $P_l(r)$ is the Legendre polynomial.

III. PARALLELIZATION OF FMM ON GPU CLUSTERS

The platform utilized in our FMM implementation is a GPU cluster, which consists of 13 computing nodes. Each node has a dual 6-core 2.66 GHz Intel Xeon processor, 48 GB RAM along with one NVidia Tesla M2090 GPU running at 1.3 GHz with 6 GB memory. The nodes are interconnected through the infiniBand interconnection. The cluster populates CUDA v7.0 and MVAPICH2 v1.8.1. (a well-known implementation of Message Passing Interface (MPI)). Two parallel programming approaches of CUDA and MPI are combined to provide the use of GPU programming across the cluster.

In this section, we provide an overview of our implementation on GPU. Figure 2 shows the main blocks which consist of pre-processing, processing and post-processing, where processes which utilize GPU are shown in solid green line, and CPU based operations are shown in dashed black line. The geometry mesh data resulting from the pre-processing step is transferred to the GPU memory, and the entire computation is performed on the GPU. The user defined results such as radar cross section, scattered fields are post-processed on the CPU.

The parallelization of the processing step in GPU cluster implementation is performed at two levels: (i) among computing nodes using MPI library, and (ii) within GPU per node using CUDA programming model. The workload of the computational task is equally distributed among the computing nodes, and the inter-node communication is minimized. This is achieved by uniformly distributing the total number of groups, M, among the n computing nodes. We only present the far interactions in this paper, since the near field and V vector calculations implementations can be found in [6]-[7].

A. Far interactions calculations

This task comprises of three calculations: radiation, and receive functions and translation matrices. The first step in the far interaction calculations is the calculation of the radiation, T^E/T^{ED} and receive, R^E, functions for Z^{far} matrix as seen in Fig. 2. It is worth noting that the radiation and the receive functions as well as the translation matrix have to be evaluated at all K directions for the unit sphere integration. The computational workload is distributed across the nodes using the group-based partitioning scheme such that M_{node} groups are allocated for each node. K evaluations for the radiation and receive functions are required for each node. The threads are grouped into blocks such that each block of threads performs N_{group} radiation/receive functions at a given group, then a total number of blocks per node

equal to $M_{node}K$.

The second task for far interactions is the calculation of the translation matrix, T_L. Similar to the calculation of the radiation/receive functions, the translation matrix has to be evaluated at all K directions for the unit sphere integration. The workload for the T_L calculations is also distributed across the nodes following the group-based technique. Each CUDA block is assigned to compute one sparse row of the T_L matrix for a given direction, and each thread computes one element in that row.

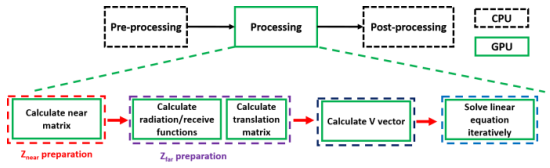


Fig. 2. FMM flow chart implementation.

B. Fast matrix-vector multiplication

The processing step is followed by solving linear equation iteratively. In this paper, the Biconjugate Gradient Stabilized method (BICGSTAB) is employed where each iteration involves the Matrix-Vector Multiplication (MVM). In this part, MVM of the far interactions are focused and the calculation of $Z_{far}I$ comprises of aggregation, translation, and disaggregation. By using the group-based scheme, the inter-node communication is required only at two steps: (i) at the beginning of the MVM to exchange the estimated values for the unknowns among the nodes, and (ii) after the aggregation step and before performing the translation step in order to update all nodes with the aggregated fields. The flow chart of the implementation is shown in Fig. 3.

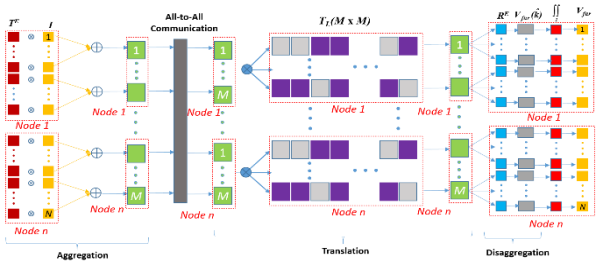


Fig. 3. Far-interaction flow chart implementation.

Each node already calculates its own portion of the radiation/receive functions, and the translation matrix. In the aggregation stage, each unknown is multiplied with its corresponding radiation functions, and is summed in each group. The aggregation stage can be performed independently for all K directions, and thus can be performed concurrently with minimal need for inter-node communication. An all-to-all communication is employed by each node to broadcast the aggregated fields to all other nodes. Then the received fields at each direction are determined by multiplying the aggregated fields with the translation matrix. Next the received fields are multiplied with the receive functions to obtain the field for each group at a given direction. Finally, an integration over the K directions of the unit sphere is performed to calculate the fields at each observed edge. The far MVM is incorporated with the near MVM to get the full ZI.

IV. EXPERIMENTAL RESULTS

A. Accuracy

To validate the accuracy of FMM implementations on GPU clusters, we calculate the radar cross section (RCS) of a 14λ diameter (254,274 unknowns) sphere with permittivity ε = 4 – 0.1j. It is illuminated by a normally incident 1 GHz x-polarized plane wave. The RCS based on our GPU implementation is compared to the results of the analytical Mie scattering. Figure 4 shows that the GPU and Mie solutions achieve a good agreement.

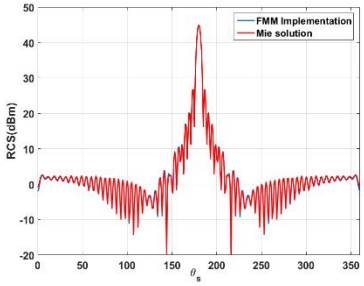


Fig. 4. RCS of a 14λ diameter dielectric sphere

B. Implementation performance on GPU cluster

Our GPU implementations is evaluated using the fixed-workload model in the first experiment. We choose a sphere diameter of 12.4λ, ~ 200K unknowns for the fixed problem size such that it demands the use of at least 8 nodes to satisfy the required memory. Speedup and scalability are used to evaluate the GPU implementations. The speed up is defined as the ratio of time required by multi-node GPU implementation with respect to the 8-node CPU implementation. Scalability is the normalized speedup of multiple nodes in reference to the speedup of 8 nodes. In our analysis, we consider the total executional time and computational time. Figure 5 shows the speedup factors and the measured time of two cases. It is observed that the computation time achieves a speedup factor of 171.5 on 8 nodes, and takes 158.9 seconds. Due to the nature of the fixed workload model, each node carries less workload when the number of computing nodes increase. Therefore the computation time decreases linearly with the increase in nodes (274.5 seconds for computation and 229.9 seconds for total). The slightly smaller speedup factors for the total time as compared to the computation time are due to the inter-node

communications for transferring the data in our GPU implementations.

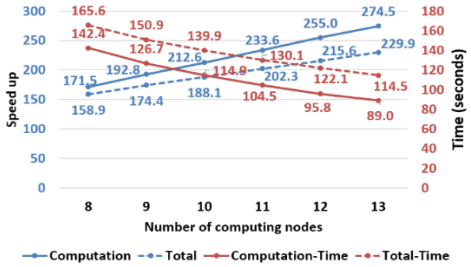


Fig. 5. Speedup analysis for the fixed-workload model (vs. 8 nodes CPU implementation, 100 iterations) Computational CPU exec time = 24,421 sec, total CPU exec time = 26,315 sec.

It is observed in Fig. 6 that both computation and total time scale closely to the theoretical linear expectation for the fixed workload problem. This good scalability demonstrates that our implementation has efficiently parallelized the algorithm and reduced the communication overhead.

The second experiment is the fixed-time model. As the problem size is increased, the number of nodes also increases, so that the GPU memory in each node is fully utilized. Our GPU implementation can process a maximum problem size of 254K unknowns with a speedup factor of 169.6 for the computation, and 137 for total execution time as shown in Fig. 7.

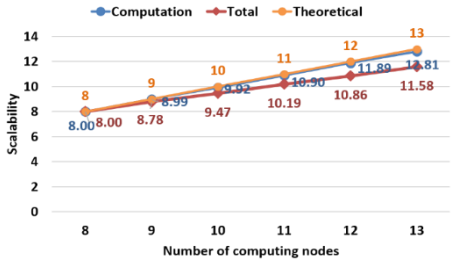
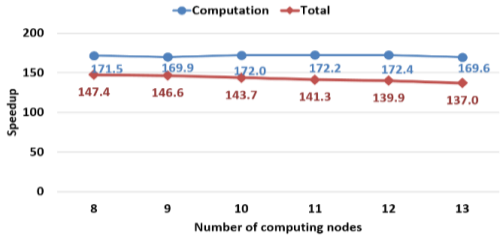


Fig. 6. Scalability analysis for the fixed-workload model.



Nodes	8	9	10	11	12	13
Size (Diameter/ unknowns)	12.4λ/ 199K	12.7λ/ 209K	13.1λ/ 222K	13.5λ/ 234K	13.8λ/ 246K	14λ/ 254K

Fig. 7. Speedup analysis when the number of nodes increases along with problem size increases (vs. multi-node CPU, 100 iterations).

VI. CONCLUSION

In this paper, the GPU implementation of FMM for dielectric electromagnetic scattering problems using our 13-node GPU cluster is demonstrated. The maximum problem size is determined by the available on-board GPU memory. For the same degree of accuracy, the GPU implementation outperforms the CPU implementation. Moreover, the GPU implementation has a good scalability as the number of computing nodes increases.

REFERENCES

[1] E. Bleszynski, M. Bleszynski, and T. Jaroszewicz, "AIM: Adaptive integral method for solving large-scale electromagnetic scattering and radiation problems," *Radio Science*, vol. 31, no. 5, pp. 1225-1251, 1996.

[2] F. X. Canning, "The impedance matrix localization (IML) method for moment-method calculations," *IEEE Antennas Propagat. Mag.*, vol. 32, no. 5, pp. 18-30, 1990.

[3] R. Coifman, V. Rokhlin, and S. Wandzura, "The fast multipole method for the wave equation: A pedestrian prescription," *IEEE Antennas Propagat. Mag.*, vol. 35, no. 3, pp. 7-12, June 1993.

[4] S. M. Rao, D. R. Wilton, and A. W. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Trans. Antennas Propag.*, vol. AP-30, no. 3. pp. 409-418, May 1982.

[5] O. Ergul and L. Gurel, "Efficient parallelization of the multilevel fast multipole algorithm for the solution of large-scale scattering problems," *IEEE Trans. Antennas Propag.*, vol. 56, no. 8, pp. 2335-2345, August 2008.

[6] Q. M. Nguyen, V. Dang, O. Kilic, and E. El-Araby, "Parallelizing fast multipole method for large-scale electromagnetic problems using GPU clusters," *Antennas and Wireless Propagation Letters, IEEE*, vol. 12, pp. 868-871, 2013.

[7] V. Dang, Q. Nguyen, and O. Kilic, "Fast multipole method for large-scale electromagnetic scattering problems on GPU cluster and FPGA-accelerated platforms," *Applied Computational Electromagnetics Society Journal*, vol. 28, no. 12, 2013.

[8] X. Q. Sheng, J.-M. Jin, J. Song, W. C. Chew, and C.-C. Lu, "Solution of combined-field integral equation using multilevel fast multipole algorithm for scattering by homogeneous bodies," *IEEE Transactions on Antennas and Propagation*, vol. 46, no. 11, pp. 1718-1726, 1998.