Parallel Implementations of Multilevel Fast Multipole Algorithm on Graphical Processing Unit Cluster for Large-scale Electromagnetics Objects

Nghia Tran and Ozlem Kilic

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

Abstract **─** This paper investigates solving large-scale electromagnetic scattering problems by using the Multi-level Fast Multipole Algorithm (MLFMA). A parallel implementation for MLFMA is performed on a 12-node Graphics Processing Unit (GPU) cluster that populates NVidia Tesla M2090 GPUs. The details of the implementations and the performance achievements in terms of accuracy, speed up, and scalability are shown and analyzed. The experimental results demonstrate that our MLFMA implementation on GPUs is much faster than (up to 37x) that of the CPU implementation.

Index Terms **─** Graphics Processing Unit (GPU), Multilevel Fast Multipole Algorithm (MLFMA).

I. INTRODUCTION

Over the past twenty years, various numerical techniques have been developed to reduce the computational time and memory requirements of full-wave electromagnetic models without significant loss of accuracy, including adaptive integral method (AIM) [1], impedance matrix localization (IML) [2], fast multipole method (FMM) [3], and multi-level fast multipole algorithm [4]. Compared with the others, MLFMA is among the most suitable techniques for large-scale problems. It reduces the computational complexity of the method of moments (MoM) from $O(N^2)$ to $O(NlogN)$, where N denotes the number of unknowns, whereas AIM, IML and FMM have the complexities of $O(N^{3/2}logN)$, $O(N^2logN)$, and $O(N^{3/2})$, respectively.

Recently, many authors have investigated the parallelization of MLFMA on CPU clusters [5] in solving problems of hundreds of thousands to millions of unknowns. In [6], CPU clusters were used to implement MLFMA using Open MP and MPI library to solve a billion unknowns. Multi-GPU implementation was also investigated on a single node, multi-GPU computer without using the MPI library [7]. In this paper, we demonstrate the implementation of MLFMA for electromagnetics problems on GPU clusters by using the MPI library.

We demonstrate the parallelization of MLFMA on a 12-node GPU cluster each of which is populated with an NVidia Tesla M2090 GPU. An MVAPICH2 implementation of MPI is used for cluster parallel programming. This paper is the continuation of our GPU implementation of FMM by using GPU clusters. In [9] and [10], GPU implementation for single level Fast Multipole Method (FMM) solves only the maximum problem size up to 656K unknowns on 13 nodes. In this paper, our MLFMA implementation on GPU cluster can solve up to 1.1 M unknowns. We demonstrate that the implementation of MLFMA on GPUs is faster than that of the CPU. The performance of the implementation is analyzed by using a PEC sphere.

The rest of the paper is organized such that Section II provides an overview of MLFMA. Section III presents the parallel implementation of MLFMA on GPU clusters. Experimental results are discussed in Section IV, followed by the conclusions in Section V.

II. OVERVIEW OF THE MULTILEVEL FAST MULTIPOLE ALGORITHM

The fundamental principles of MLFMA and its applications in electromagnetics have been studied in literature [4]-[5]. In this section, we provide a brief overview to help our discussion on its parallel implementation, which is presented in Section III.

MLFMA was invented based on the grouping concept to accelerate the iterative solution of the linear equation system ZI = V of the Method of Moment (MoM), where I represents the unknown currents, V depends on the incident field, and Z is the impedance matrix. The main idea of the grouping concept is shown in Fig. 1, where the M edges in the mesh of a given structure are categorized into an N-level tree structure connecting groups of different sizes from the finest (level N) to the coarsest level (level 0). Based on the groups' proximity, the impedance matrix Z can be split into two matrices, Z^{near} and Z^{far}, corresponding to near and far interactions as shown in Equation (1):

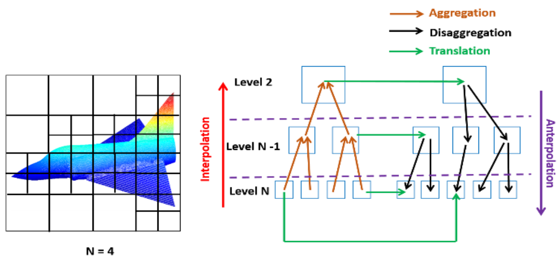
$$\sum_{m'}^{M} Z_{mm'} I_{m'} = \sum_{m'}^{M} Z_{mm'}^{near} I_{m'} + \sum_{m'}^{M} Z_{mm'}^{far} I_{m'} = V_m , \qquad (1)$$

where m and m' are observation and source edges in the

Submitted On: February 27, 2016
Accepted On: May 10, 2016

mesh, respectively.

The Z^{near} matrix comprises of interactions between edges in spatially nearby groups, and is computed and stored using the conventional MoM [8]. During the iterative solution, the near matrix is calculated by the regular sparse matrix-vector multiplications (MVMs). The remaining edges, whose parents are near, constitute the far term as shown in Fig. 1 (b). By treating the interactions between the edges that are spatially far-away using MLFMA, Z^{far} matrix does not need to be explicitly computed and stored. Instead, the far components can benefit from the fast MVMs during the iterative solution. The Z_{far} matrix is factorized into radiation, receive and translation functions, as explained in [4].



(a) (b)

Fig. 1. MLFMA general concepts: (a) concept of the MLFMA tree, and (b) MLFMA concept of far interactions.

The far component is calculated through five main stages: aggregation, translation, and disaggregation, interpolation and anterpolation as shown in Fig. 1 (b).

In the aggregation stage, radiated fields among the groups from level N (the finest level) to level 2 are calculated. At the finest level N, the radiation functions for a group are computed by combining the radiation patterns of the basic function of all edges in this group. From level N-1 to level 2, the radiation functions for each group are computed from the combination of the radiation function of its children group of the finer level using shifting and interpolation.

In the disaggregation stage, the receive functions at each group are computed from level 2 to level N by combining the local incoming waves due to translation and the incoming waves from parent groups of the coarser level using shifting and anterpolation.

The translation stage is identical to FMM [3], and the details of interpolation and anterpolation can be found in [5].

III. PARALLELIZATION OF MLFMA ON GPU CLUSTERS

In this section, we provide an overview of our implementation on GPU. The implementation consists of pre-processing, processing and post-processing. 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 CPU.

The GPU cluster used for our implementation consists of 12 computing nodes. Each node has a dual 6-core 2.66 GHz Intel Xeon processor, 48GB RAM along with one NVidia Tesla M2090 GPU running at 1.3 GHz supported with 6GB of GPU memory. The nodes are interconnected through the InfiniBand interconnection. The cluster populates CUDA v6.0 and MVAPICH2 v1.8.1 (a well-known implementation of Message Passing Interface (MPI)).

In the processing step, 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. The parallelization of the GPU cluster implementation is performed at two levels: (i) among the computing nodes using MPI library, and (ii) within the GPU per node using CUDA programming model. Within each node, the CUDA thread-block model is utilized to calculate the workload assigned to that node. We only present the far interactions in this paper, since the near field and V vector calculations implementations can be found in [9]-[10].

All CUDA kernels are implemented to calculate Z^{near} matrix, and far interactions which includes the radiation/receive functions, translation matrix, and interpolation/anterpolation matrices. In fast matrix-vector multiplication (MVM), CUDA kernel is also utilized to compute the radiated fields, translation fields and received fields in the aggregation, translation and disaggregation stages, respectively. MPI library is also used to gather results from each node in the end of MVM stage.

A. Far interactions calculations

This task comprises of five calculations: radiation, and receive functions, interpolation, anterpolation and translation matrices.

(i) Radiation and Receive Function Calculations

The first step in the far interaction calculations is the calculation of the radiation, T^E, and receive, R^E, functions for Z^{far} matrix. They are complex conjugates of each other. Thus their implementations are similar. Following the M group distribution, each node handles the calculations of K directions for M_{node} groups. Given this amount of workload per node, the CUDA kernel is launched with $M_{node}.K$ blocks such that each block implements M_{group} radiation/receive function calculations at a given direction, resulting in a total of $M_{node}.K$ blocks per node.

(ii) Translation Matrix Calculation

The second task for far interactions is the calculation of the translation matrix, T_L. The workload for the T_L calculations is also distributed across the nodes following the group-based technique. By careful investigations, allocating a CUDA block on a single row of the matrix is the efficient way for the translation matrix calculation to save memory requirements. 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.

(iii) Interpolation and Anterpolation Matrices

The third task for the far interactions is the calculation of interpolation and anterpolation matrices. They are transposes of each other. Thus their implementation is similar. Each node handles the calculations of $K_{children/node}$ rows of $K_{children}$xK_{parent} interpolation matrix, where $K_{children}$ is number of directions of a finer level, and K_{parent} is number of directions of a coarser level. The CUDA kernel is launched with $K_{children/node}$ blocks per node. In each block, the maximum number of threads (1024 threads) are utilized in order to implement the full number of K_{parent} directions.

B. Fast matrix-vector multiplication

The next stage for the processing is the solution for the linear system where we employ the iterative method known as the biconjugate gradient stabilized method (BiCGSTAB). The calculation of $Z^{far}I$ comprises of five stages: aggregation, translation, interpolation, anterpolation and disaggregation, as shown in Fig. 2. Using a group-based partitioning technique, the unknown current vector I (N_{edges}x1) is distributed across the computing nodes on GPU clusters.

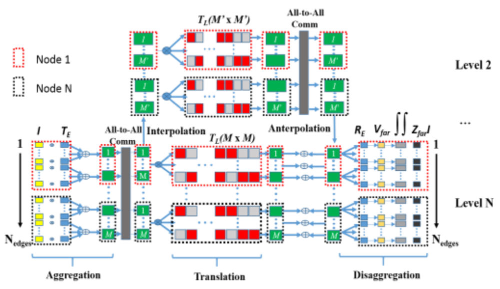


Fig. 2. Far matrix-vector-multiplication in parallel.

In the aggregation stage, at level N, each node computes the radiated fields for M_{node} groups for K directions by multiplying the unknowns I with their corresponding radiation functions, T^E, and accumulating within each group. After the aggregation step, an all-to-all communication is employed by each node to broadcast the radiated fields to all other nodes. The radiated fields from level N-1 to level 2 are computed by multiplying interpolation matrices with radiated fields of children groups at lower levels.

In the translation, the radiated fields at each direction are calculated from the sum of the multiplication of the translation matrix and the radiated fields, and the received fields from parent groups at upper levels using anterpolation.

In the disaggregation stage, the received fields of all M group at level N are multiplied with the corresponding receive functions, and integrated over the partitioned K directions of the unit sphere. The far components of MVM are then incorporated with the near components of MVM. At the end of MVM, the partial results from all nodes are summed together and all nodes are updated.

IV. EXPERIMENTAL RESULTS

A. Accuracy

First, we verify the accuracy of our GPU implementation by calculating the radar cross section (RCS) of a 9λ diameter (corresponding to 0.27 m and 100,000 unknowns) perfect electrically conducting (PEC) sphere illuminated by an 1 GHz x-polarized normally incident field. The results are compared to Mie scattering. It can be observed in Fig. 3 that the GPU results and the analytical solutions show a very good agreement.

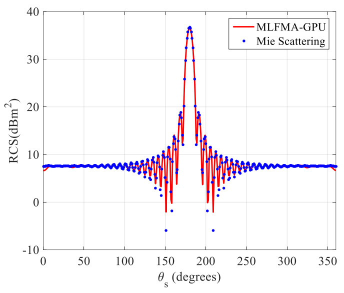


Fig. 3. RCS of a 9λ diameter PEC sphere.

B. Implementation performance on GPU cluster

In the first experiment, our GPU implementation is evaluated using the fixed-workload model (Amdahl's Law). A 22.4λ diameter PEC sphere (650K unknowns) is chosen such that it demands the use of at least 7 nodes to satisfy the required memory. Two metrics are used for the performance evaluation: speed up and scalability. The speed up is defined as the ratio of time required by multi-node GPU implementation with respect to the 7-node CPU implementation. Scalability is the normalized speedup of multiple nodes in reference to the speedup of 7 nodes. As shown in Fig. 4, the speedup factor increases from 23.7 for 7 nodes to 37 for 12 nodes. Since each node processes less workload, the GPU execution time decreases as the number of nodes increases. The inter-node communication overhead results in the difference between the speedup of total execution time and computation time. For 7 computing nodes, the speed-up for the near-field system matrix is over 86 (CPU computation time: 848s, GPU computation time: 9.5s), while the speed-up of the BICGstab iterative solution is over 22 times for 100 iterations, which is restricted by the overhead communication between computing nodes (CPU computation time: 9100s, GPU computation time: 415.1s).

In order to investigate the scalability of this implementation, we compare how the speedup improves with increasing number of computing nodes as we keep the problem size constant, as observed in Fig. 5. The computation speedup scales similar to the theoretical linear behavior, demonstrating our efficient hardware implementation. The total speedup scales closely to the theoretical expectation demonstrating our efficiency in reducing the inter-node communication overhead.

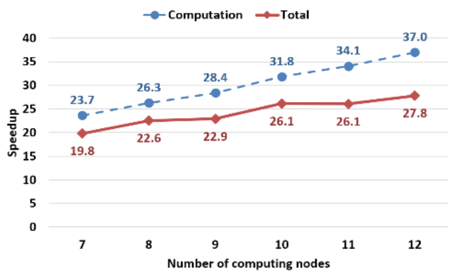


Fig. 4. Speedup analysis for the fixed-workload model (vs. 7 nodes CPU implementation, 100 iterations). Computational CPU exec time = 5573 sec, total CPU exec time = 5627 sec.

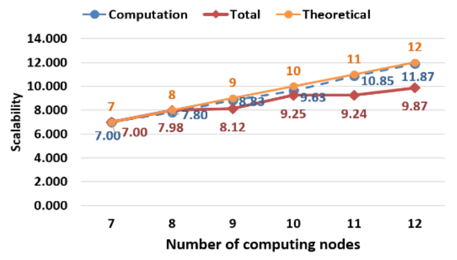
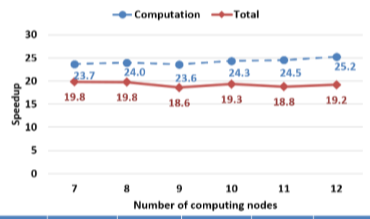


Fig. 5. Scalability analysis for the fixed-workload model.

In the second experiment, we investigate the largest problem size our GPU implementation can handle. As the number of nodes increases, the problem size is also increased so that the GPU memory in each node in fully utilized. As shown in Fig. 6, the GPU implementation can process a maximum problem size of 1.1 M unknowns with a speed up factor of 25.2.



Nodes	7	8	9	10	11	12
Size (Radius/Unknowns	11.2λ/0.65M	12.1λ/0.7M	12.5λ/0.81M	13λ/0.88M	13.4λ/0.93M	13.9λ/1.1M

Fig. 6. 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 MLFMA for electromagnetic scattering problems up to 1.1 million unknowns using our 12-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 Ant. Prop. 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] J. M. Song and W. C. Chew, "Multilevel fast multipole algorithm for solving combined field integral equations of electromagnetic scattering," *Microw. Opt. Tech. Lett.*, vol. 10, pp. 14-19, Sep. 1995.

[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, Aug. 2008.

[6] X.-M. Pan, W.-C. Pi, M.-L. Yang, Z. Peng, and X.-Q. Sheng, "Solving problems with over one billion unknowns by the MLFMA," *Antennas and Propaga. IEEE Trans. on*, vol. 60, no. 5, pp. 2571-2574, 2012.

[7] J. Guan, S. Yan, and J.-M. Jin, "An OpenMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems," *Antennas and Propaga., IEEE Trans. on*, vol. 61, no. 7, pp. 3607-3616, 2013.

[8] 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.

[9] 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.

[10] 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.