

© 2013 Jian Guan

OPENMP-CUDA IMPLEMENTATION OF THE MOMENT METHOD  
AND MULTILEVEL FAST MULTIPOLE ALGORITHM ON  
MULTI-GPU COMPUTING SYSTEMS

BY

JIAN GUAN

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Electrical and Computer Engineering  
in the Graduate College of the  
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Adviser:

Professor Jianming Jin

# ABSTRACT

In this thesis, the method of moments (MoM) and the multilevel fast multipole algorithm (MLFMA) are implemented for GPU computation based on the hybrid OpenMP-CUDA parallel programming model. The resultant algorithms are called the OpenMP-CUDA-MoM and the OpenMP-CUDA-MLFMA, respectively. Both of the proposed methods are applied to compute electromagnetic scattering by a three-dimensional conducting object.

For the OpenMP-CUDA-MoM, the multi-GPU parallelization of system matrix assembly, iterative solution, and fast evaluation of radar cross section (RCS) are discussed in detail. The parallel efficiency versus number of devices is investigated through the calculation of a conducting sphere on different number of GPUs. The parallel efficiency of the total computation is over 87%. The total speedup for the monostatic RCS calculation of a NASA almond by 4 GPUs is between 80 and 260 times.

For the GPU accelerated MLFMA, the hierarchical parallelization strategy is employed, which ensures a high computational throughput for the GPU calculation. The resulting OpenMP-based multi-GPU implementation is capable of solving real-life problems with over 1 million unknowns with a remarkable speedup. The RCS of a few benchmark objects are calculated to demonstrate the accuracy of the solution. The results are compared with those from the CPU-based MLFMA and measurements. The capability of the proposed method is analyzed through the examples of a sphere, an aircraft and a missile-like object. The total speedup achieved by 4 GPUs is between 20 and 80 times.

*To my parents, for their love and support*

# ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my advisor, Professor Jian-Ming Jin, who has provided me much guidance, support, help, and encouragement, without which I would not have been able to complete this thesis. All I have learned during the study and research under Professor Jin's direction will greatly benefit my future work.

Besides my advisor, I would like to thank every member in Professor Jin's group for their helpful discussions. My special appreciation is given to Dr. Su Yan for his great help and suggestions on my research.

Finally, I want to express appreciation to my parents for their dedication and their continued love and support.

# TABLE OF CONTENTS

LIST OF TABLES . . . . .	vii
LIST OF FIGURES . . . . .	viii
CHAPTER 1 INTRODUCTION . . . . .	1
CHAPTER 2 AN OUTLINE OF MOM AND MLFMA FOR ELEC- TROMAGNETIC SCATTERING . . . . .	4
2.1 Integral Equations of Electromagnetic Scattering . . . . .	4
2.2 Geometrical Modeling and Domain Discretization . . . . .	5
2.3 Curvilinear RWG Basis Functions . . . . .	6
2.4 Solution of Integral Equations . . . . .	7
2.5 Figures . . . . .	10
CHAPTER 3 HYBRID OPENMP-CUDA PARALLEL PROGRAM- MING MODEL AND GPU ARCHITECTURE . . . . .	12
3.1 Hybrid OpenMP-CUDA Parallel Programming Model . . . . .	13
3.2 GPU/CUDA Architecture . . . . .	13
3.3 Computational Efficiency and Parallel Efficiency . . . . .	15
3.4 Figures . . . . .	16
CHAPTER 4 MULTI-GPU PARALLELIZATION OF MOM . . . . .	18
4.1 System Matrix Assembly . . . . .	19
4.2 Iterative Solution . . . . .	19
4.3 RCS Evaluation . . . . .	20
4.4 Numerical Analysis . . . . .	21
4.5 Summary . . . . .	23
4.6 Figures . . . . .	24
4.7 Tables . . . . .	28
CHAPTER 5 MULTI-GPU PARALLELIZATION OF MLFMA . . . . .	30
5.1 Near-Field System Matrix Assembly . . . . .	30
5.2 Parallelization on Far-Field Interaction . . . . .	31
5.3 Multi-GPU Implementation Using Pinned Memory . . . . .	32
5.4 Numerical Analysis . . . . .	33

5.5	Summary . . . . .	36
5.6	Figures . . . . .	38
5.7	Tables . . . . .	46
CHAPTER 6 CONCLUSION . . . . .		48
REFERENCES . . . . .		50

# LIST OF TABLES

4.1	Speedup of system matrix assembly of a PEC sphere with diameter of $4\lambda$ (single GPU) . . . . .	28
4.2	Speedup of BiCGstab solution of a PEC sphere with diameter of $4\lambda$ (single GPU) . . . . .	28
4.3	Speedup of RCS calculation of a PEC sphere with diameter of $4\lambda$ (single GPU) . . . . .	28
4.4	Speedup of total computation of a PEC sphere with diameter of $4\lambda$ (single GPU) . . . . .	29
4.5	Speedup of monostatic RCS calculation of a NASA almond at 3 GHz (CFIE, 4 GPUs) . . . . .	29
4.6	Speedup of monostatic RCS calculation of a NASA almond at 9 GHz (CFIE, 4 GPUs) . . . . .	29
5.1	Speedup of the global memory strategy for bistatic RCS calculation of a missile-like object at 3 GHz . . . . .	46
5.2	Speedup of the pinned memory strategy for bistatic RCS calculation of a missile-like object at 3 GHz . . . . .	46
5.3	Speedup of the bistatic RCS calculation of a PEC sphere with diameter of $30\lambda$ . . . . .	47
5.4	Speedup of the bistatic RCS calculation of an aircraft at 1.5 GHz . . . . .	47

# LIST OF FIGURES

2.1	The sketch of a curvilinear triangular element. (a) The curvilinear triangle in the rectangular coordinate system. (b) The curvilinear triangle in the parametric coordinate system. . . . .	10
2.2	The definition of a CRWG basis function. . . . .	10
2.3	The process of the far-field interaction. $\mathbf{S}_i$ and $\mathbf{B}_i$ stand for the radiation pattern and receiving pattern at level $i$ respectively. . . . .	11
3.1	Speedup achieved for a matrix-matrix multiplication using NVIDIA Quadro FX5800 graphics card versus Intel's Xeon W3520 CPU (2.66 GHz). . . . .	16
3.2	A typical OpenMP-CUDA parallel programming model. . . . .	16
3.3	A typical CUDA-capable NVIDIA GPU architecture [1, 2]. . . . .	17
3.4	Memory accessibility of the thread divisions [1, 2]. . . . .	17
4.1	System matrix assembly in the OpenMP-CUDA-MoM. (a) Data fetch from basis stream. (b) Data fetch from testing stream. . . . .	24
4.2	BiCGstab solution in the OpenMP-CUDA-MoM. . . . .	25
4.3	HH-polarized bistatic RCS for the PEC sphere with diameter of $4\lambda$ . . . . .	25
4.4	Parallel efficiency of the OpenMP-CUDA-MoM versus number of devices . . . . .	26
4.5	HH-polarized monostatic RCS for the NASA almond at 3 GHz. . . . .	26
4.6	HH-polarized monostatic RCS for the NASA almond at 9 GHz. . . . .	27
5.1	Near-field system matrix assembly. (a) Pattern of near-field system matrix. (b) Process of matrix filling. . . . .	38
5.2	Thread allocation for the aggregation phase at the (L-2)th level. . . . .	38
5.3	Implementation of far-field interaction on Multi-GPU. (a) Parallel scheme for aggregation. (b) Parallel scheme for translation. (c) Parallel scheme for disaggregation. . . . .	39

5.4	Scattering analysis of a cone-sphere with a gap at 3 GHz. The total length of this object is 0.689 m. (a) The HH-polarized monostatic RCS in the $xz$ -plane. (b) Real part of the current density with the incidence angle $\theta = 0^\circ$ and $\phi = 0^\circ$ (in linear scale). . . . .	40
5.5	Scattering analysis of a NASA almond at 9 GHz. The size of this object is $25.24 \text{ cm} \times 9.75 \text{ cm} \times 3.25 \text{ cm}$ . (a) The HH-polarized monostatic RCS in the $xy$ -plane. (b) Real part of the current density with the incidence angle $\theta = 90^\circ$ and $\phi = 180^\circ$ (in linear scale). . . . .	41
5.6	Scattering analysis of a missile-like object. The length of the body is 3 m, and the thickness of the wing is 1 cm. A 3 GHz plane wave is incident from the angle $\theta = 0^\circ$ and $\phi = 0^\circ$ . (a) The HH-polarized bistatic RCS in the $xz$ -plane. (b) Real part of the current density induced on the surface of the scatterer (in linear scale). . . . .	42
5.7	Scattering analysis of the PEC spheres with diameters of $4\lambda$ , $6\lambda$ , $12\lambda$ , and $30\lambda$ . (a) The four devices speedup of the OpenMP-CUDA-MLFMA versus the number of unknowns (unknowns = 18162, 41316, 158333, 1063155). (b) The HH-polarized bistatic RCS of the $30\lambda$ PEC sphere. . . . .	43
5.8	Scattering analysis of the aircraft at frequencies of 200 MHz, 400 MHz, 780 MHz and 1.5 GHz. (a) The four devices speedup of the OpenMP-CUDA-MLFMA versus the number of unknowns (unknowns = 20319, 70413, 269859, 1001946). (b) The VV-polarized bistatic RCS in the $yz$ -plane at 1.5 GHz. . . . .	44
5.9	Real part of the current density at 1.5 GHz with the incidence angle $\theta = 60^\circ$ and $\phi = 270^\circ$ (in linear scale). . . . .	45

# CHAPTER 1

## INTRODUCTION

Numerical methods for electromagnetic analysis have been developed rapidly since the 1960s. Many well-known numerical methods have been introduced, including the finite element method (FEM), the finite-difference time-domain method (FDTD), and the method of moments (MoM) [3]. Compared with the FEM and FDTD, the MoM can avoid the truncation errors introduced by absorbing boundary conditions and the numerical dispersion errors due to the discretization of propagation spaces. These properties make MoM an ideal method for solving electromagnetic radiation and scattering problems. However, the major disadvantage of the MoM is that it has  $O(N^2)$  computational and storage complexities, which result in a large memory requirement and a tremendous amount of computation time. To accelerate the computation and reduce the memory requirement, the multilevel fast multipole algorithm (MLFMA) has been developed and widely used in electromagnetic scattering analysis due to its  $O(N\log N)$  computational complexity [3, 4].

Even with a near optimal computational complexity, the computational cost of the MLFMA is still prohibitively high when it is used for large electromagnetic problems. In practical application, many of those problems are required to be solved within a very short time. In order to further accelerate the computation, parallel computation has been applied to the traditional MLFMA [5–9] to take advantage of computer hardware advancement. In 2005, a hybrid parallel MLFMA based on the distributed memory system using the message passing interface (MPI) was proposed [6]. The strategy is rather straightforward. For the finer levels in MLFMA, the groups at the same level are partitioned into different processors and each processor gets approximately the same number of groups. For the coarser levels, the far-field patterns (FFPs) at the same level are partitioned equally among all processors and all groups are replicated for every processor. However, when the number of processors increases, this parallel strategy does not

work well around the transition level where neither the number of groups nor the number of FFPs is large enough to get a good parallel efficiency. To alleviate this problem, a hierarchical partitioning strategy was later proposed by simultaneously partitioning the groups and their FFPs at all levels [8]. More recently, a hybrid MPI-OpenMP-MLFMA method was implemented based on the hybrid shared/distributed memory architecture to solve the problems with over one billion unknowns [9].

All the preceding parallel strategies are implemented using the CPU parallel programming models such as MPI and OpenMP. Recently the graphics processing unit (GPU), which is basically a many-core computing system, has received more and more attention from computational electromagnetics (CEM) community due to its low price and high computational throughput [1, 2]. There has been intensive research dedicated to developing the GPU parallelized algorithms. The differential-equation-based methods such as the finite-difference time-domain (FDTD) and the discontinuous Galerkin finite element (DGTD) methods have been implemented on GPUs [10–12]. The integral-equation-based methods such as the method of moments (MoM) and the time-domain integral equation (TDIE) method have also been accelerated by GPUs [13–16]. Besides, many GPU-incorporated fast algorithms for efficient evaluation of electromagnetic fields have been presented [17–19], such as the non-uniform grid interpolation method (NGIM), the box-based adaptive integral method (B-AIM), the multilevel plane-wave time-domain (PWTD) method, and the fast multiple method (FMM). For the GPU-accelerated MLFMA, the CUDA (compute unified device architecture) implementation of low-frequency MLFMA on a single GPU was proposed with the essential idea of “one thread per observer” [20]. The observer stands for the parent group in the aggregation phase, the child group in the disaggregation phase, and the destination group in the translation phase, respectively. However, that implementation strategy results in a low parallel efficiency when the number of groups decreases at coarse levels.

In order to improve the parallel efficiency and solve large problems, this thesis first proposes a multi-GPU accelerated MoM, called the OpenMP-CUDA-MoM, which is developed by hybridizing the OpenMP and the CUDA parallel programming models. To be specific, the parallelization of system matrix assembly, iterative solution, and RCS calculation on multi-GPU computing systems are discussed in detail. To solve larger problems and

further speedup the computation, an OpenMP-CUDA based implementation of MLFMA, called OpenMP-CUDA-MLFMA, is proposed. For the computation of far-field interaction, the groups and the FFPs are parallelized hierarchically. A global memory strategy and a pinned memory strategy are proposed for different application situations. This algorithm is shown to have a high computational efficiency when solving large electromagnetic scattering problems.

The remainder of the thesis is organized as follows. In Chapter 2, the formulations and implementations of MoM and MLFMA are outlined, followed by an introduction to the hybrid OpenMP-CUDA parallel programming model and GPU/CUDA architecture in Chapter 3. In Chapter 4, the multi-GPU parallelization of MoM is presented. Then we discuss the OpenMP-CUDA-MLFMA in Chapter 5. The conclusion is drawn in Chapter 6.

## CHAPTER 2

# AN OUTLINE OF MOM AND MLFMA FOR ELECTROMAGNETIC SCATTERING

In order to present the implementation strategies of the OpenMP-CUDA-MoM and the OpenMP-CUDA-MLFMA clearly, it is necessary to have a brief review of the formulations and their numerical implementations. In this chapter, the integral equations of electromagnetic scattering are presented. In order to numerically solve those integral equations, the geometrical modeling, domain discretization, and the curvilinear Rao-Wilton-Glisson (CRWG) basis functions are first discussed in detail. Then the MoM is applied to translate the integral equation into a system of linear equations. Finally, the basic idea and numerical implementation of MLFMA are presented to speed up the matrix-vector products and reduce the memory requirement.

### 2.1 Integral Equations of Electromagnetic Scattering

Consider a three-dimensional (3D) conducting object illuminated by an incident field  $(\mathbf{E}^i, \mathbf{H}^i)$ . The electric-field integral equation (EFIE) and the magnetic-field integral equation (MFIE) are given by

$$\eta \mathcal{T}(\mathbf{J}) = -\hat{\mathbf{n}} \times \mathbf{E}^i(\mathbf{r}) \quad \mathbf{r} \in S \quad (2.1)$$

$$-\frac{1}{2}\mathbf{J} + \mathcal{K}(\mathbf{J}) = -\hat{\mathbf{n}} \times \mathbf{H}^i(\mathbf{r}) \quad \mathbf{r} \in S \quad (2.2)$$

respectively, where  $\mathbf{J}$  denotes the unknown surface current density and the integral operators  $\mathcal{T}$  and  $\mathcal{K}$  are defined as

$$\mathcal{T}(\mathbf{J}) = ik\hat{\mathbf{n}} \times \int_S \left( \mathcal{I} + \frac{\nabla \nabla}{k^2} \right) \frac{e^{ikR}}{4\pi R} \cdot \mathbf{J}(\mathbf{r}') d\mathbf{r}' \quad (2.3)$$

$$\mathcal{K}(\mathbf{J}) = \hat{\mathbf{n}} \times P.V. \int_S \nabla \frac{e^{ikR}}{4\pi R} \times \mathbf{J}(\mathbf{r}') d\mathbf{r}' \quad (2.4)$$

where *P.V.* stands for the Cauchy principal value integration,  $S$  denotes the surface of the conducting object,  $k$  and  $\eta$  denote the free-space wavenumber and impedance,  $\hat{\mathbf{n}}$  is the outwardly directed normal unit vector,  $\mathcal{I}$  represents the identity operator, and  $R = |\mathbf{r} - \mathbf{r}'|$  denotes the distance between the field and source points.

Both the EFIE and the MFIE can be solved for  $\mathbf{J}$ . However, for a given closed surface, both of them will suffer from the problem of interior resonance at certain frequencies when the exterior medium is lossless [3]. To eliminate this problem, we can combine (2.1) and (2.2) together to form the combined-field integral equation (CFIE) which is given by

$$\begin{aligned} \alpha \hat{\mathbf{n}} \times \eta \mathcal{I}(\mathbf{J}) + (1 - \alpha) \eta \left[ \frac{1}{2} \mathbf{J} - \mathcal{K}(\mathbf{J}) \right] \\ = -\alpha \hat{\mathbf{n}} \times [\hat{\mathbf{n}} \times \mathbf{E}^i(\mathbf{r})] + (1 - \alpha) \eta \hat{\mathbf{n}} \times \mathbf{H}^i(\mathbf{r}) \quad \mathbf{r} \in S \end{aligned} \quad (2.5)$$

where  $\alpha \in [0, 1]$  is the combination parameter. The numerical method to solve the EFIE, MFIE, and CFIE is discussed specifically in the following sections.

## 2.2 Geometrical Modeling and Domain Discretization

In order to solve the unknown surface current density from integral equations, the geometry of the object needs to be described mathematically, which is called geometrical modeling. The quality of the geometrical modeling will directly affect the accuracy of the numerical solution. Because it is impossible to find the basis functions defined on the entire solution domain, the object is usually discretized into subdomains which are referred as elements. The subdomain basis functions are employed to provide an approximation of the unknown solution within an element.

The curved surface of an object can be modeled by curvilinear triangular elements. As shown in Figure 2.1, a curvilinear triangle can be defined by six nodes, three of which are the vertices of the triangle, and the other three are the midpoints of the three curved edges. A curvilinear triangle in the rectangular coordinate system can be mapped onto a triangle in a parametric

coordinate system using the coordinate transformation as

$$\mathbf{r}(\xi_1, \xi_2) = \sum_{j=1}^6 \varphi_j(\xi_1, \xi_2, \xi_3) \mathbf{r}_j \quad (2.6)$$

where  $\mathbf{r}_j$  are the rectangular coordinates of the six controlling nodes in Figure 2.1(a),  $\xi_1, \xi_2, \xi_3$  are the parametric coordinates, which satisfy the relation  $\xi_1 + \xi_2 + \xi_3 = 1$ . Hence only two of these three are independent. The shape functions  $\varphi_j$  are defined in the parametric coordinates

$$\varphi_1 = \xi_1(2\xi_1 - 1) \quad (2.7)$$

$$\varphi_2 = \xi_2(2\xi_2 - 1) \quad (2.8)$$

$$\varphi_3 = \xi_3(2\xi_3 - 1) \quad (2.9)$$

$$\varphi_4 = 4\xi_1\xi_2 \quad (2.10)$$

$$\varphi_5 = 4\xi_2\xi_3 \quad (2.11)$$

$$\varphi_6 = 4\xi_1\xi_3. \quad (2.12)$$

By using the presented curvilinear triangular elements, the arbitrarily curved surfaces can be modeled with good accuracy and high flexibility.

## 2.3 Curvilinear RWG Basis Functions

After the surface discretization of the object using curvilinear triangular elements, the CRWG basis functions can be defined on these triangular elements, as shown in Figure 2.2. The definition of the CRWG basis functions can be expressed mathematically as

$$\mathbf{f}_n(\mathbf{r}) = \begin{cases} \mathbf{\Lambda}^+ & \mathbf{r} \in T_n^+ \\ -\mathbf{\Lambda}^- & \mathbf{r} \in T_n^- \\ 0 & \text{otherwise.} \end{cases} \quad (2.13)$$

Here,  $\mathbf{\Lambda}^+$  and  $\mathbf{\Lambda}^-$  are the basis functions related with the edge  $n$ , defined in the triangles  $T_n^+$  and  $T_n^-$  which share the common edge  $n$  ( $n$  is the global index of the basis function). The  $\mathbf{\Lambda}^+$  and  $\mathbf{\Lambda}^-$  can be represented locally using the basis function  $\mathbf{\Lambda}_i$  defined on the edge  $l_i$  of the curvilinear triangular element

shown in Figure 2.1, where  $l_i$  is the edge which is opposite to the  $i$ th vertex of the triangle ( $i$  is the local index of the vertex in the triangle). The expression of  $\mathbf{\Lambda}_i$  can be written as

$$\mathbf{\Lambda}_1 = \frac{1}{J}(\xi_2 \mathbf{l}_3 - \xi_3 \mathbf{l}_2) \quad (2.14)$$

$$\mathbf{\Lambda}_2 = \frac{1}{J}(\xi_3 \mathbf{l}_1 - \xi_1 \mathbf{l}_3) \quad (2.15)$$

$$\mathbf{\Lambda}_3 = \frac{1}{J}(\xi_1 \mathbf{l}_2 - \xi_2 \mathbf{l}_1) \quad (2.16)$$

where  $J$  is the Jacobian

$$J = \left| \frac{\partial \mathbf{r}}{\partial \xi_1} \times \frac{\partial \mathbf{r}}{\partial \xi_2} \right| \quad (2.17)$$

and  $\mathbf{l}_i (i = 1, 2, 3)$  are edge vectors defined as

$$\mathbf{l}_1 = -\frac{\partial \mathbf{r}}{\partial \xi_2}, \quad \mathbf{l}_2 = \frac{\partial \mathbf{r}}{\partial \xi_1}, \quad \mathbf{l}_3 = \mathbf{l}_1 + \mathbf{l}_2. \quad (2.18)$$

The divergence of the CRWG basis function can be derived as

$$\nabla \cdot \mathbf{\Lambda}_i = \frac{2}{J}, \quad i = 1, 2, 3. \quad (2.19)$$

The charge density represented by the divergence of the CRWG basis function has the same magnitude but the opposite signs over the adjacent triangle pair, which means no artificial charges accumulated on the pair of triangles.

## 2.4 Solution of Integral Equations

The EFIE, MFIE, and CFIE can be solved by numerical methods. One of the most commonly used methods for solving integral equations is the method of moments (MoM) introduced by R. F. Harrington in 1968 [21]. Since both the EFIE and the MFIE can be considered as the special case of the CFIE, the MoM solution of the CFIE is considered here. In order to numerically solve the CFIE, the unknown current density can be expanded as

$$\mathbf{J} = \sum_{n=1}^N I_n \mathbf{f}_n \quad (2.20)$$

where  $N$  is the number of unknowns,  $\mathbf{f}_n$  denotes the vector basis functions, and  $I_n$  is the expansion coefficient yet to be determined. In this thesis, the CRWG functions [22, 23] are used as the basis functions. The application of Galerkin's method to (2.5) results in a system of linear equations

$$\sum_{n=1}^N Z_{mn} I_n = V_m \quad m = 1, 2, \dots, N \quad (2.21)$$

in which

$$Z_{mn} = \alpha \int_S \mathbf{f}_m \cdot [\hat{\mathbf{n}} \times \eta \mathcal{T}(\mathbf{f}_n)] d\mathbf{r} \quad (2.22)$$

$$+ (1 - \alpha) \eta \int_S \mathbf{f}_m \cdot \left[ \frac{1}{2} \mathbf{f}_n - \mathcal{K}(\mathbf{f}_n) \right] d\mathbf{r}$$

$$V_m = \int_S [\alpha \mathbf{E}^i + (1 - \alpha) \eta \hat{\mathbf{n}} \times \mathbf{H}^i] \cdot \mathbf{f}_m d\mathbf{r}. \quad (2.23)$$

In the MoM, the system matrix  $\mathbf{Z}$  is a full matrix, which results in  $O(N^2)$  memory requirement and computational complexity when (2.21) is solved iteratively. To speed up the matrix-vector products and reduce the memory requirement, MLFMA is extensively applied to MoM [4]. The basic idea of the MLFMA is to decompose the computation of matrix-vector products into the near-field and far-field interactions. To achieve such a decomposition, the entire object is first enclosed by a large cubic box, then divided into non-empty subcubes called groups, each subcube is further subdivided into small cubes recursively until the length of non-empty cubes at the finest level is about  $0.25\lambda$  to  $0.5\lambda$ . After the decomposition, the system of linear equations can be written as

$$\mathbf{Z}_{\text{near}} \mathbf{I} + \mathbf{Z}_{\text{far}} \mathbf{I} = \mathbf{V} \quad (2.24)$$

in which  $\mathbf{Z}_{\text{near}}$  is a block matrix, and each block represents the interaction between the testing functions in a group at the finest level and the basis functions in the same group or a neighboring group.  $\mathbf{Z}_{\text{far}}$  is the remaining part of the MoM matrix which represents the interaction between groups that are well separated [6]. The  $\mathbf{Z}_{\text{near}}$  can be calculated directly using the ordinary MoM at the finest level, and the computation of  $\mathbf{Z}_{\text{far}} \mathbf{I}$  can be done in three phases called aggregation, translation, and disaggregation. Figure 2.3 shows the basic idea for the computation of the far-field interaction,

where  $\mathbf{S}_i$  and  $\mathbf{B}_i$  stand for the radiation pattern and receiving pattern at level  $i$ , respectively. The uphill process represents the aggregation phase, the downhill process is the disaggregation phase, and the transverse process stands for the translation phase. In the aggregation phase, the fields radiated by the sources  $\mathbf{f}_n I_n$  in each group at the finest level are first projected into the spectrum space to obtain the radiation pattern, which is then aggregated to the center of the parent group at the parent level. This procedure is executed repeatedly until it reaches the coarsest level. It is easy to notice that the number of the groups becomes smaller while the size of the spectrum sampling becomes larger in the aggregation phase. In the disaggregation phase, the receiving patterns at each level comes from two sources: one is the translation of the radiation patterns at the same level; the other is the disaggregation of the receiving pattern from the parent level. Thus, the translation and disaggregation can be executed concurrently as follows [6,24]. At the coarsest level, the radiation pattern is first translated to the receiving pattern for each group, which is then distributed to the centers of child groups at the child level. At the same time, the radiation pattern at the child level is translated to the receiving pattern at the same level. Then the total receiving pattern can be achieved at the child level by summing up the above two receiving patterns. After the total receiving pattern is achieved, the next level's disaggregation and translation can be processed. This procedure is executed recursively until it reaches the finest level.

## 2.5 Figures

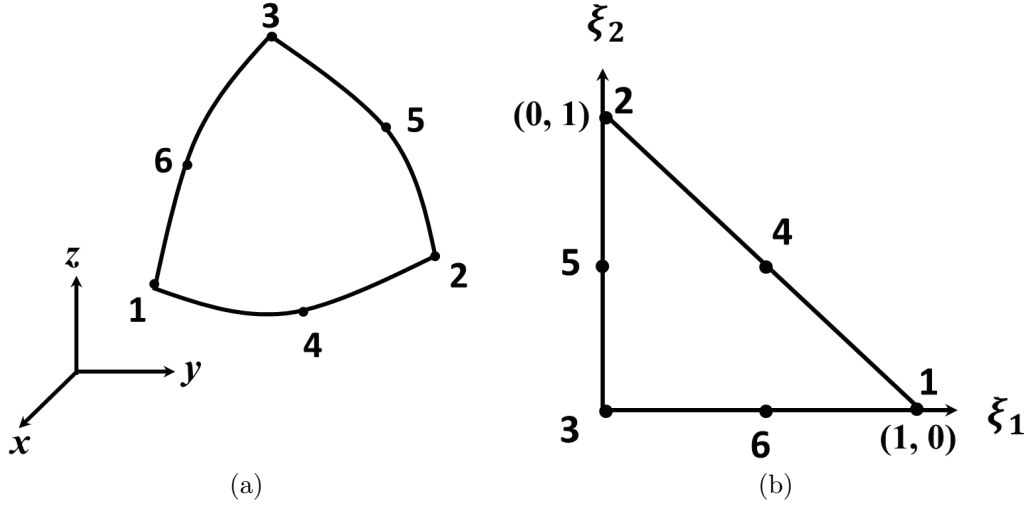


Figure 2.1: The sketch of a curvilinear triangular element. (a) The curvilinear triangle in the rectangular coordinate system. (b) The curvilinear triangle in the parametric coordinate system.

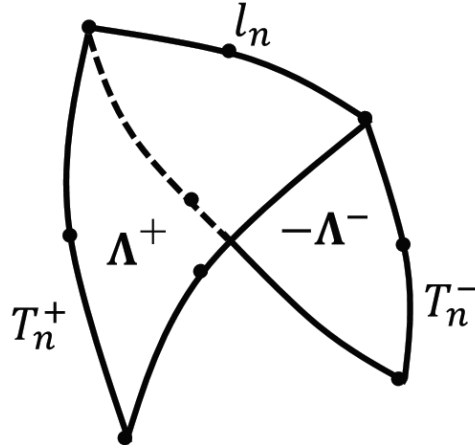


Figure 2.2: The definition of a CRWG basis function.

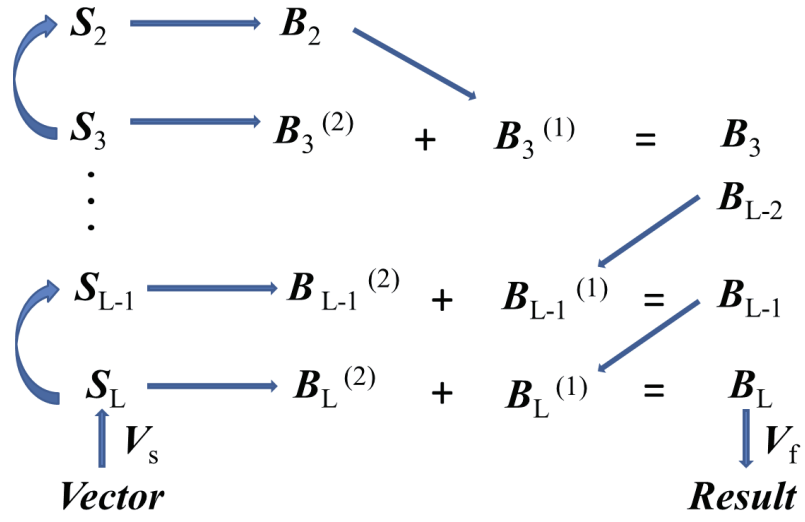


Figure 2.3: The process of the far-field interaction.  $S_i$  and  $B_i$  stand for the radiation pattern and receiving pattern at level  $i$  respectively.

# CHAPTER 3

## HYBRID OPENMP-CUDA PARALLEL PROGRAMMING MODEL AND GPU ARCHITECTURE

Parallel computing is an important technique to accelerate electromagnetic computation due to the fast development of advanced microprocessors in which multiple threads cooperate to complete the work faster. Many computational methods for electromagnetics have been parallelized on distributed and shared memory systems using the message passing interface (MPI) and OpenMP, respectively. Besides the well-developed shared memory multi-core CPUs and distributed memory cluster architectures, the graphics processing unit (GPU), as one of many-cores computing system, has received more and more attention from the scientific computing community due to its low price and high computational efficiency. Compared with multi-core computing systems, GPU focuses more on the execution throughput of parallel computing which can be illustrated in terms of floating-point calculation and memory bandwidth. As an example to demonstrate the significant computational horsepower, Figure 3.1 shows the speedup achieved for a matrix-matrix multiplication using NVIDIA Quadro FX5800 graphics card versus Intel's Xeon W3520 CPU (2.66 GHz), in which the speedups achieved by using global memory and shared memory are over 100 and 1000 times, respectively.

Before the OpenMP-CUDA-MoM and the OpenMP-CUDA-MLFMA are presented in detail, we first review the key features of the hybrid OpenMP-CUDA parallel programming model and the GPU/CUDA architecture. Then, we define the computational efficiency and parallel efficiency to describe the performance of GPU computation.

### 3.1 Hybrid OpenMP-CUDA Parallel Programming Model

Multi-GPU parallelization can be developed using hybrid parallel programming techniques. The most popular approaches to implement multi-GPU application are using the MPI-CUDA model and the OpenMP-CUDA model. In this section, we introduce the multi-GPU implementation using the OpenMP-CUDA model. The OpenMP programming model is based on the shared memory multi-core CPUs architecture [25], and CUDA is developed for the shared memory many-core GPUs architecture [2]. A typical OpenMP-CUDA programming model is shown in Figure 3.2. In general the program consists of one or more phases [2, 25]. The serial phase of the program is first executed by the master thread on the host (CPU). Then multiple GPUs labeled as devices take over the work in the parallel phase. Specifically, multiple CPU worker threads are allocated by an OpenMP instruction, and each worker thread manages one device. The data-parallel functions called kernels are executed on each device. When a kernel is launched, a large number of GPU threads are generated to exploit data parallelism. Those threads are organized into a two-dimensional (2D) grid of blocks, with each block built by a 2D or 3D array of threads. All of those threads generated by the kernel will carry out the same instructions during the parallel phase. With enough threads in a kernel to execute the same code simultaneously, the latency hiding mechanism [1, 2, 20] can be fully utilized to make the parallelism highly efficient. After every device finishes its parallel computation, the CPU will pick up the runtime and execute the instructions in the serial phase.

### 3.2 GPU/CUDA Architecture

It should be emphasized that the significant computational efficiency of GPU results from its specific hierarchical architecture and excellent memory bandwidth. To elaborate the GPU acceleration of the MoM/MLFMA, the understanding of the hardware architecture of GPUs is necessary. Figure 3.3 shows the architecture of a typical CUDA-capable NVIDIA GPU [1, 2], which is organized into an array of streaming multiprocessors (SMs). Off the chip, all the SMs in one device share a very high bandwidth memory called global

memory and a high speed read-only memory called constant memory. The lifetime of variables in the global memory and constant memory is the entire application unless they are freed by the programmer. On the chip, each SM contains a number of streaming processors (SPs) which share control logic, cache, and shared memory. Each SP has its own small number of registers which usually store the private and frequently accessed variables because they can be accessed very fast.

For the CUDA software architecture, the method of allocating different types of memory on the GPU varies by the hierarchy of threads in the kernel. As shown in Figure 3.4, multiple threads form a thread block and multiple thread blocks form a grid. Registers and shared memory are on-chip memories which can be read and written with short latency. Each thread owns a number of registers which can be accessed by the respective thread. Registers usually store the private and frequently accessed variables. Shared memory is allocated to thread blocks. The threads in a block can access variables stored in shared memory which is allocated to the block. The cooperation between threads in a block is taken through the shared memory. As for the grid, the global memory and constant memory are allocated to the grid of threads. Before a kernel is invoked, the programmer needs to allocate memory on the device and transfer the data from host memory to device memory. After a kernel finishes the calculation, the programmer needs to copy back the results to the host memory and free the device memory.

Usually the size of device memory and on-chip memories are not enough to solve large problems. One remedy is to use multi-GPU, and the other is to use pinned memory. Pinned is a special host memory, which is also called page-locked memory. One important property of this memory is that the operating system guarantees pinned memory never be paged out to disk. Besides, the pinned memory has approximately twice the performance of the standard pageable memory when it is used for transferring data between the host and device. However, the transfer speed is restricted by the peripheral component interconnect express (PCIe) transfer speed and the system front-side bus speed [26]. Therefore, the full utilization of the hierarchical memory and the reduction of data communication are crucial in GPU computation.

Recognizing that the GPUs are well suited in dealing with massive data parallelism and weak at executing with logical instructions while the CPUs are optimized for sequential instruction performance, one should expect that

CEM codes execute the numerically intensive parts on the GPUs and the sequential parts on the CPU. A well-investigated coordinating strategy can make the GPU-incorporated algorithm much more efficient than the purely CPU implemented algorithm.

### 3.3 Computational Efficiency and Parallel Efficiency

In order to describe the performance of GPU computation, the computational efficiency and parallel efficiency are defined. Let  $T_c$  and  $T_g$  be the time for CPU and GPU computation respectively. The computational efficiency  $s$  is defined as

$$s = \frac{T_c}{T_g}. \quad (3.1)$$

The computational efficiency is the speedup using GPU versus CPU, which is simply referred to efficiency.

The parallel efficiency is defined as the speedup using multi-GPU versus single GPU. Let  $p$  be the number of devices, and let  $T_1$  be the time for single GPU computation and  $T_p$  be the time for  $p$  GPUs computation. Then, the parallel efficiency  $\eta$  is defined as

$$\eta = \frac{T_1}{T_p \times p} \times 100. \quad (3.2)$$

The ideal parallel efficiency is 100%. Usually the parallel efficiency is referred as the efficiency versus number of devices.

### 3.4 Figures

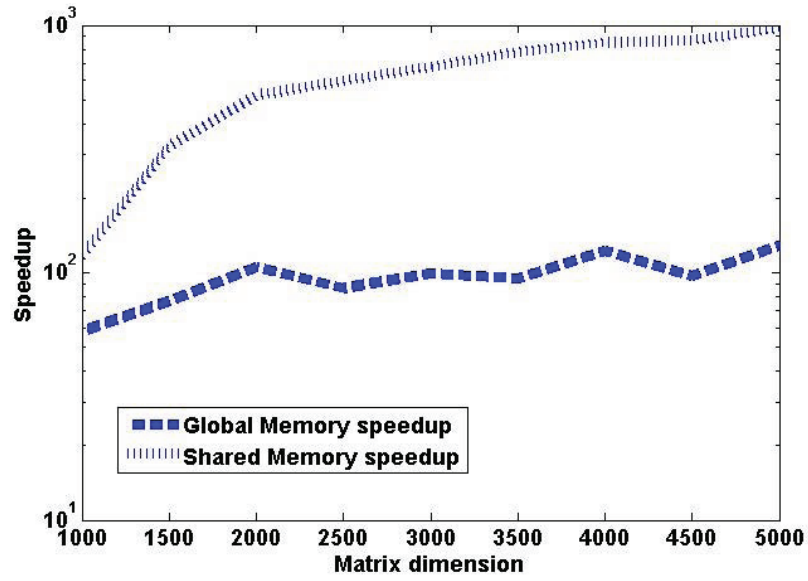


Figure 3.1: Speedup achieved for a matrix-matrix multiplication using NVIDIA Quadro FX5800 graphics card versus Intel's Xeon W3520 CPU (2.66 GHz).

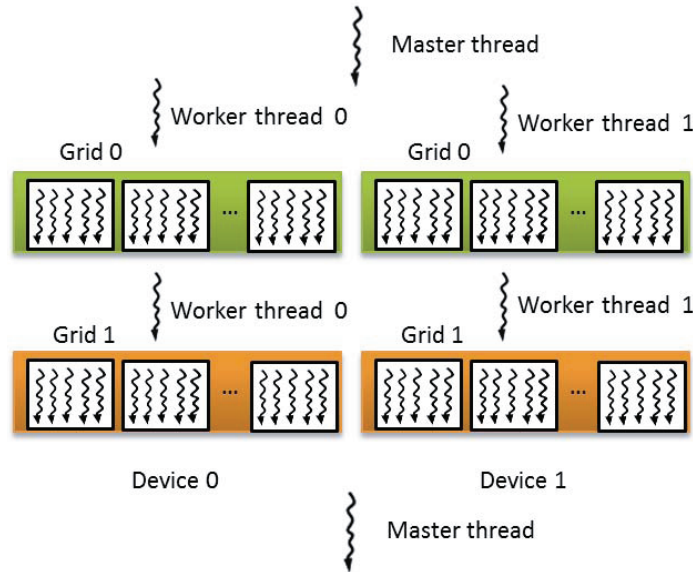


Figure 3.2: A typical OpenMP-CUDA parallel programming model.

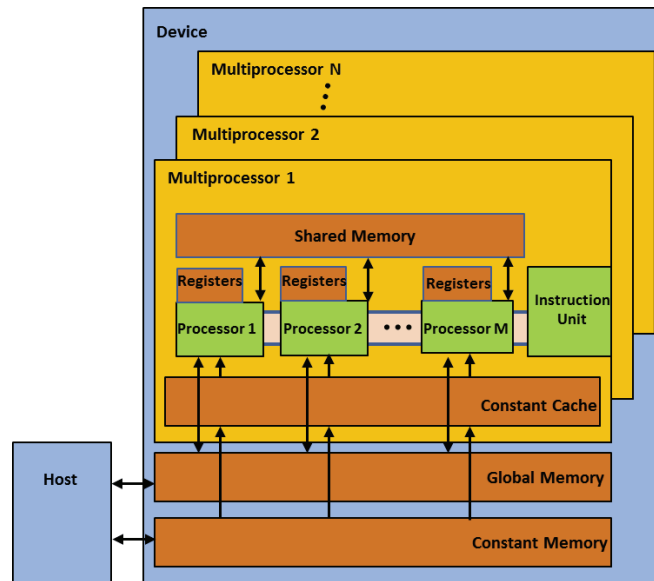


Figure 3.3: A typical CUDA-capable NVIDIA GPU architecture [1,2].

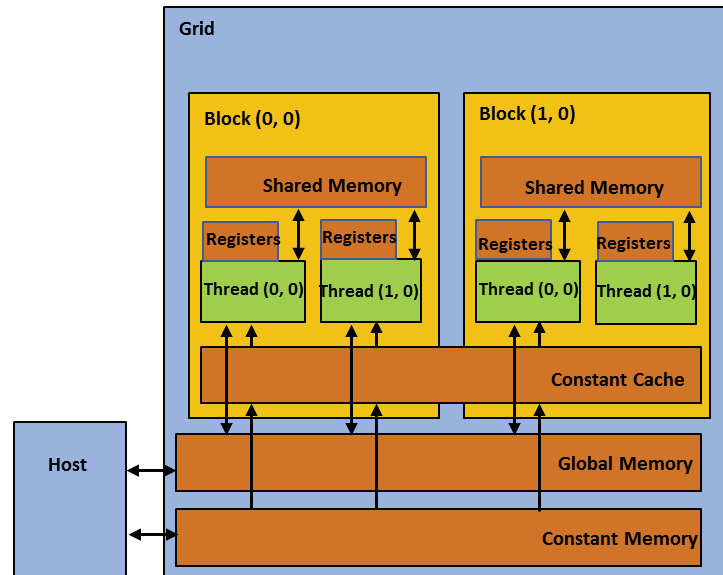


Figure 3.4: Memory accessibility of the thread divisions [1,2].

# CHAPTER 4

## MULTI-GPU PARALLELIZATION OF MOM

The MoM has been developed and widely used for solving electromagnetic radiation and scattering problems due to the following merits: the surface discretization results in fewer unknowns compared with the volume discretization; the Sommerfeld radiation condition is automatically satisfied without an additional absorbing boundary. However, the major disadvantage of the MoM is that it has  $O(N^2)$  computational and storage complexities, which result in a large memory requirement and a tremendous amount of computation time. Thus, it is critical for practical application to speed up the computation and reduce the memory requirement. In order to take the advantage of GPU parallel computing, GPU parallelization of the MoM has been investigated in the computational electromagnetics community. A GPU accelerated MoM using Brook was first proposed in [13]. Later on, there has been intensive research focusing on the GPU implementation of MoM using CUDA [14–16]. To the best of our knowledge, all the research is focused on single GPU implementation, and the multi-GPU implementation of MoM using OpenMP-CUDA parallel programming model has not been published in literatures. In this chapter, a multi-GPU implementation of MoM is proposed. We first present the parallelization of system matrix assembly. Then we propose a multi-GPU accelerated iterative solution. Moreover, fast evaluation of radar cross section (RCS) using multi-GPU is detailed. In order to investigate the numerical accuracy and computational efficiency of the OpenMP-CUDA-MoM, the RCS of a few benchmark problems are evaluated in the section of numerical analysis.

## 4.1 System Matrix Assembly

The most time-consuming part in the 3D MoM is the system matrix assembly. To accelerate MoM using GPU, replacing this intensive computation part with GPU calculation is the first consideration. Because storing a system matrix requires  $O(N^2)$  memory, and hence a single GPU cannot provide sufficient memory when problems are very large, therefore, the multi-GPU strategy is considered to remedy this problem.

In order to illustrate the process of system matrix assembly, we consider 2 GPUs here. Figure 4.1 shows the process of system matrix assembly on 2 GPUs. First the system matrix is equally divided into the two devices, with each device getting the approximately same number of rows to calculate. Then we use the OpenMP technique to allocate 2 CPU threads which are labeled by ID indices. By calling the function `cudaSetDevice(ID)`, each device is managed by the corresponding CPU thread, and all the CUDA based instructions are executed on the corresponding device. On each device, the massive number of threads are assigned to assemble a portion of the system matrix. Specifically, a 2D grid of threads is generated on the device according to the size of the submatrix to be filled, and each thread calculates one element in the submatrix. The work for each thread is first to fetch the essential data such as the geometry data from the basis stream and testing stream on the global memory, then to calculate the system element by following the standard steps of the MoM, and finally to store the results back into the global memory. Since each device keeps a portion of the system matrix locally, there is no data communication between the host and device or the device and device in the procedure described. It is easy to see that the calculation of each matrix element is independent, which leads to a high parallel efficiency regardless of the geometrical shape of the object.

## 4.2 Iterative Solution

Accelerating the solution of the system equation will further speed up the MoM application. Both the direct and iterative solutions are considered for the GPU acceleration. For the direct solution, CUDA Lapack (CULA) is used on GPU. However, the direct solver has an  $O(N^3)$  computational complexity;

hence, cannot be applied to solve large problems. In order to solve large electromagnetic problems, we mainly discuss the multi-GPU implementation of the iterative solution in this section.

The matrix-vector multiplication (MVM) is the most time-consuming step in the iterative solution. Figure 4.2 shows the procedure of MVM using multi-GPU. Holding a portion of the system matrix and the whole source array, each device calculates MVM separately to obtain two arrays, which are then combined to a whole array on CPU. After that, the whole array is transferred to each GPU for the next MVM. For example, suppose we have two devices available and the problem to be solved has  $2N$  unknowns. Hence, the size of the system matrix is  $2N \times 2N$ , and the length of the source array is  $2N$ . Each device stores a submatrix with a size of  $N \times 2N$  and the whole source array with a length of  $2N$ . After a MVM is finished, each device obtains an array with a length of  $N$ . Those two arrays are transferred to fill in an array with a length of  $2N$  on CPU. During this procedure, the OpenMP thread synchronization is used to guarantee that all devices have finished copying before they execute the next step. After the whole array has been constructed on CPU, the array is transferred from the host to each device. So far one MVM is completed. Based on the preceding discussion, it is easy to see that the acceleration of an iterative solution is limited by the thread synchronization and the data communications between the host and device. However, the time of data transfer is proportional to  $O(N)$ , while the computational complexity of MVM is  $O(N^2)$ ; hence, the speedup of the iterative solution should become larger when solving larger problems, because in that case the MVM will take a larger portion of the execution time in the iterative solution.

### 4.3 RCS Evaluation

Fast evaluation of RCS is critical to the application of electromagnetic scattering such as target tracking. In this section, the multi-GPU accelerated RCS evaluation is presented. Because basis functions can be regarded as current sources, by which the scattering field is radiated, a 1D grid of threads is allocated and each of them is related with one basis function. The work for each thread is to calculate the scattering field radiated by the corresponding

basis function. The RCS at a specific observation angle can be obtained by the superposition of those scattering fields. In order to avoid the thread writing conflict on GPU, the superposition is completed on CPU. To further accelerate the RCS evaluation, the OpenMP parallel technique is employed to generate multiple CPU threads, and each CPU thread manages one device to calculate the RCS at one specific observation angle. Because the calculations of RCS at different observation angles are independent of each other, the RCS evaluation can be parallelized effectively.

## 4.4 Numerical Analysis

In this section, a conducting sphere and a NASA almond are solved to demonstrate the numerical accuracy and computational efficiency of the OpenMP-CUDA-MoM. The CRWG [22, 23] functions are used as the basis and testing functions to discretize the EFIE, MFIE, and CFIE ( $\alpha = 0.5$ ). All the numerical examples are solved by the biconjugate gradient stabilized method (BiCGStab) with a targeted relative residual error of  $10^{-3}$ . The single-precision floating-point arithmetic is used. The CPU-MoM and the OpenMP-CUDA-MoM are executed respectively on a single CPU (Intel Xeon W3520) using one thread and a 4-GPU system equipped with 4 Nvidia Tesla C2050 GPUs.

### 4.4.1 Scattering by a Conducting Sphere

The problem of scattering by a perfect electrically conducting (PEC) sphere with a diameter of  $4\lambda$  is first considered. The object is illuminated by a plane wave with a frequency of 300 MHz, and is discretized into 17820 unknowns.

The single GPU acceleration of the system matrix assembly, iterative solution, and RCS calculation is first considered. As shown in Figure 4.3, the bistatic RCS calculated by the CPU-MoM and the OpenMP-CUDA-MoM agree well with the Mie series solution. Table 4.1 shows the speedup of the system matrix assembly, and it can be seen that the MFIE has the highest speedup among the three integral equations, while the EFIE has the lowest acceleration, which results from the different treatments of singularity. The Duffy transformation is performed to evaluate the singular integral in

the EFIE; while in the MFIE, the singular points are simply bypassed in the Gaussian quadrature. Hence the thread divergence of the EFIE is larger than the one of the MFIE. For the acceleration of solution, the high performance CUDA Basic Linear Algebra Subprograms (BLAS) is utilized in the iterative method. As shown in Table 4.2, the speedup achieved in the BiCGstab solution is between 70 and 130. For the RCS calculation, 35 times speedup is achieved by using a single GPU, which is shown in Table 4.3. Table 4.4 shows that over 45 times total speedup is achieved for the EFIE, MFIE, and CFIE.

The preceding acceleration is based on a single GPU. For the multi-GPU acceleration, the parallel efficiency of the OpenMP-CUDA-MoM is shown in Figure 4.4. It is found that the parallel efficiency is over 87% for the total computation. For a different number of devices, the system matrix assembly and RCS calculation have the parallel efficiency over 95%. The parallel efficiency of the BiCGstab solution becomes lower as the number of devices grows, which is due to the thread synchronization and data communications between the host and devices.

#### 4.4.2 Scattering by a NASA Almond

The second benchmark object is a NASA almond with a size of  $25.24 \text{ cm} \times 9.75 \text{ cm} \times 3.25 \text{ cm}$ . The object is illuminated by 3-GHz and 9-GHz plane waves, respectively. For the 3-GHz case, the object is discretized into 3984 curvilinear triangular patches, leading to 5976 unknowns. The monostatic RCS calculated by the CPU-MoM and the OpenMP-CUDA-MoM agree well with the measurement, as shown in Figure 4.5. Table 4.5 shows the speedup of the computation at 3 GHz. The speedup of the system matrix assembly is 190 times. The monostatic RCS at 181 angles are calculated, leading to 82 times speedup. The total speedup achieved is over 80 times. For the 9-GHz case, the object is discretized into 11134 curvilinear triangular patches, resulting in 16701 unknowns. As shown in Figure 4.6, the results from the CPU-MoM and the OpenMP-CUDA-MoM agree well with the measured data. The total speedup achieved is over 260 times.

## 4.5 Summary

In this chapter, an OpenMP-CUDA based implementation of the MoM is presented for computing wave scattering problems of 3D conducting objects on multi-GPU computing systems. The multi-GPU parallel strategies of the system matrix assembly, iterative solution, and RCS evaluation are discussed in detail. To demonstrate the numerical accuracy and computational efficiency of the OpenMP-CUDA-MoM, the electromagnetic scattering of a PEC sphere and a NASA almond are simulated respectively. The RCS results show that the accuracy of the proposed method is guaranteed. The efficiency versus the number of devices is investigated through the computation of a PEC sphere. The system matrix assembly and RCS evaluation have the parallel efficiency over 95%, which is significant. The parallel efficiency of the iterative solution is not as good as the one of the matrix assembly and RCS evaluation, which is due to the thread synchronization and the data communications between the host and devices. For the total computation, the parallel efficiency is over 84%. The total speedup for the monostatic RCS calculation of a NASA almond by 4 GPUs is between 80 and 260.

## 4.6 Figures

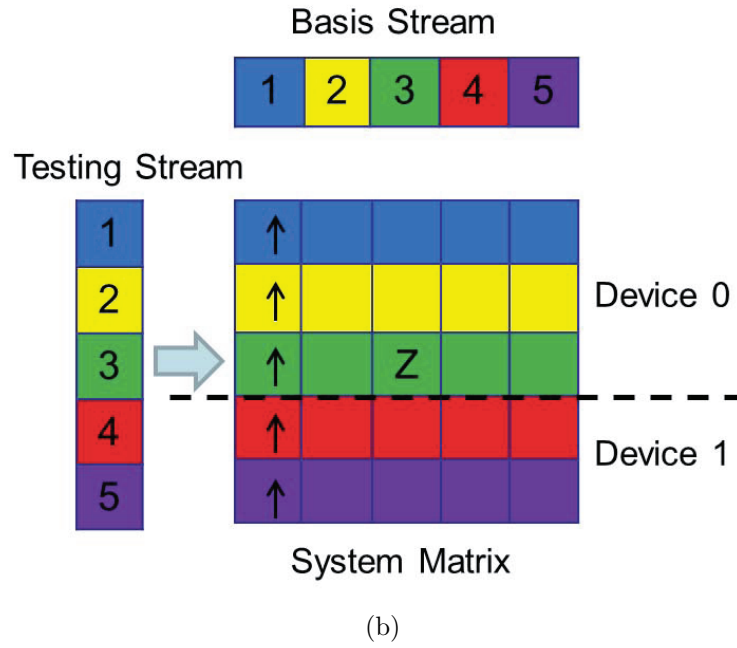
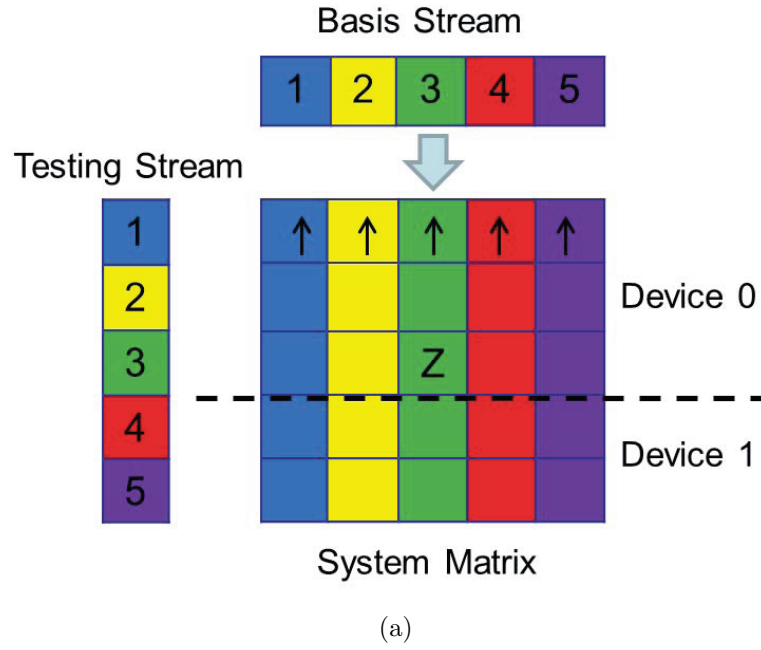


Figure 4.1: System matrix assembly in the OpenMP-CUDA-MoM. (a) Data fetch from basis stream. (b) Data fetch from testing stream.

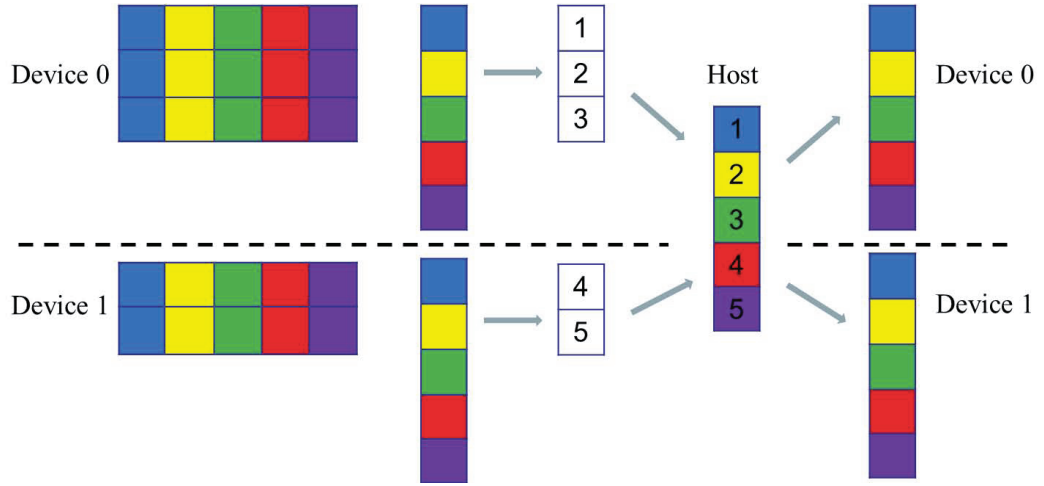


Figure 4.2: BiCGstab solution in the OpenMP-CUDA-MoM.

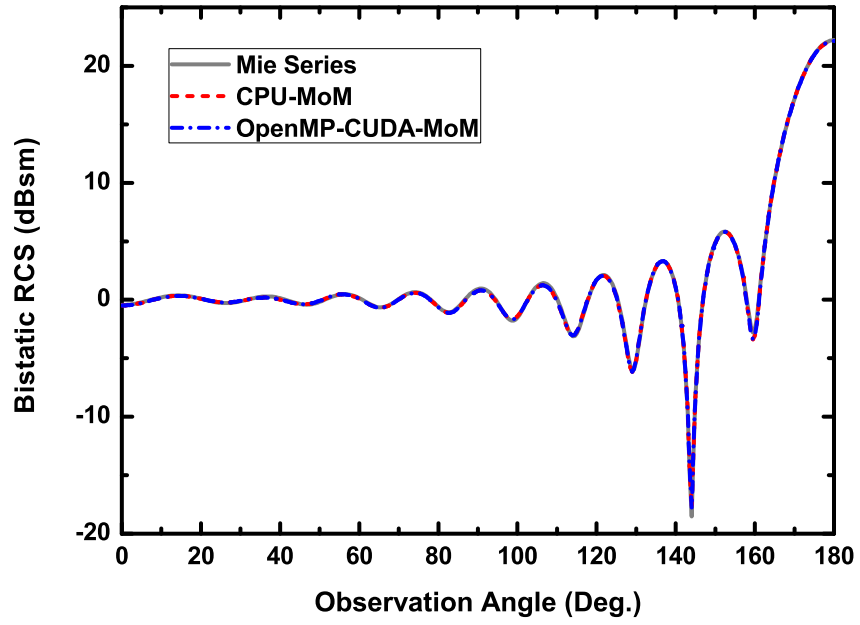


Figure 4.3: HH-polarized bistatic RCS for the PEC sphere with diameter of  $4\lambda$ .

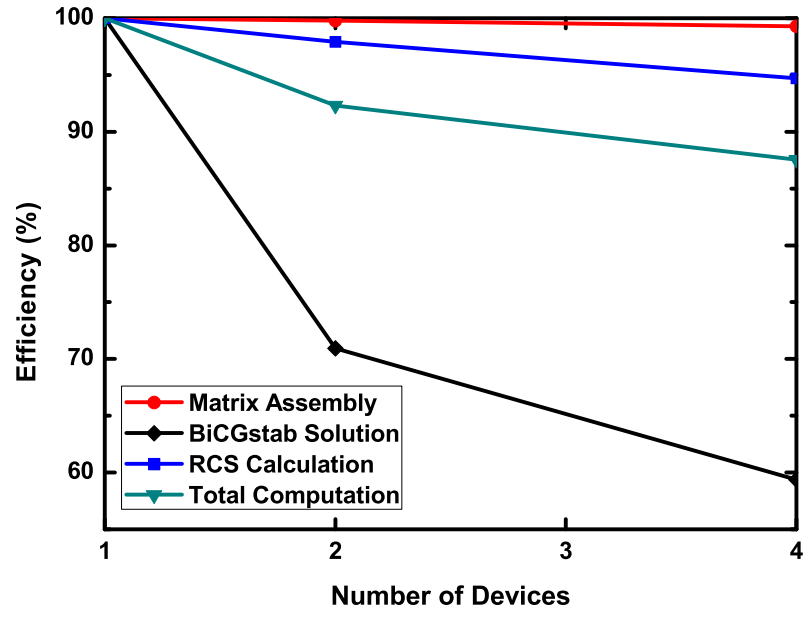


Figure 4.4: Parallel efficiency of the OpenMP-CUDA-MoM versus number of devices

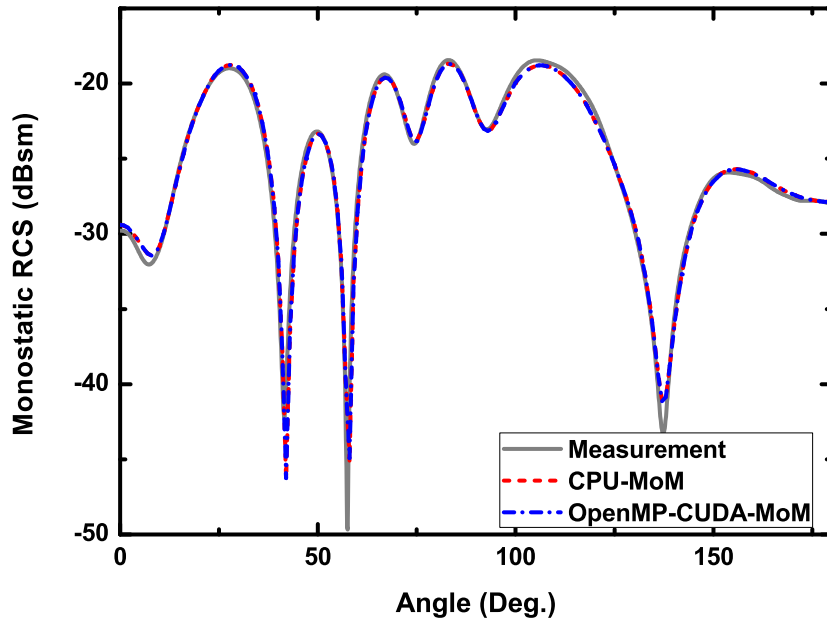


Figure 4.5: HH-polarized monostatic RCS for the NASA almond at 3 GHz.

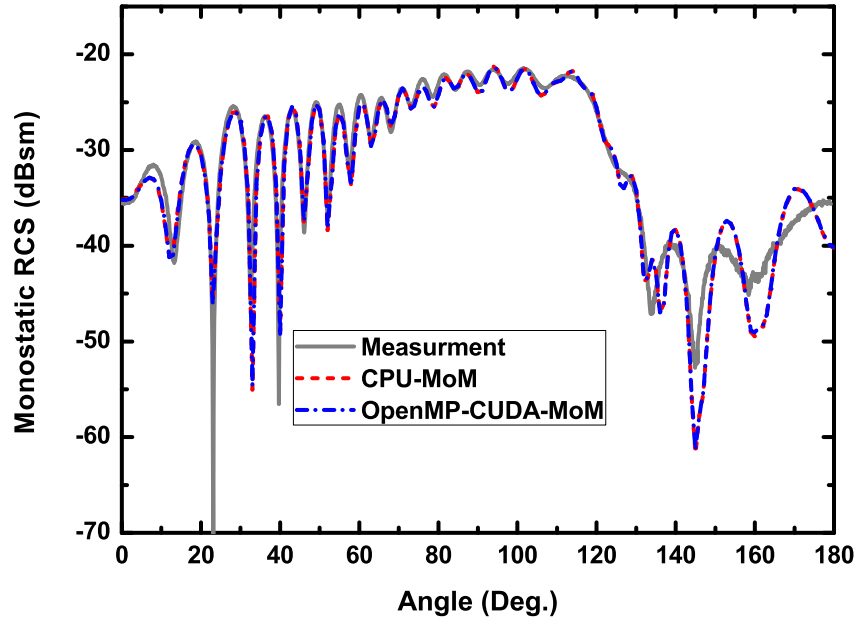


Figure 4.6: HH-polarized monostatic RCS for the NASA almond at 9 GHz.

## 4.7 Tables

Table 4.1: Speedup of system matrix assembly of a PEC sphere with diameter of  $4\lambda$  (single GPU)

	CPU-MoM (sec.)	OpenMP- CUDA-MoM (sec.)	Speedup
EFIE	1364	40	34.1
MFIE	3021	57	53.0
CFIE	4427	98	45.2

Table 4.2: Speedup of BiCGstab solution of a PEC sphere with diameter of  $4\lambda$  (single GPU)

	CPU-MoM (sec.)	OpenMP- CUDA-MoM (sec.)	Speedup
EFIE	1470	10	147.0
MFIE	130	1	130.0
CFIE	146	2	73.00

Table 4.3: Speedup of RCS calculation of a PEC sphere with diameter of  $4\lambda$  (single GPU)

	CPU-MoM (sec.)	OpenMP- CUDA-MoM (sec.)	Speedup
Bistatic RCS	35	1	35.0

Table 4.4: Speedup of total computation of a PEC sphere with diameter of  $4\lambda$  (single GPU)

	CPU-MoM (sec.)	OpenMP- CUDA-MoM (sec.)	Speedup
EFIE	2869	51	56.3
MFIE	3186	59	54.0
CFIE	4608	101	45.6

Table 4.5: Speedup of monostatic RCS calculation of a NASA almond at 3 GHz (CFIE, 4 GPUs)

	CPU-MoM (sec.)	OpenMP- CUDA-MoM (sec.)	Speedup
System Matrix	381	2	190.5
Monostatic RCS	27882	338	82.5
Total Computation	28263	350	80.8

Table 4.6: Speedup of monostatic RCS calculation of a NASA almond at 9 GHz (CFIE, 4 GPUs)

	CPU-MoM (sec.)	OpenMP- CUDA-MoM (sec.)	Speedup
System Matrix	2972	21	141.5
Monostatic RCS	118610	435	272.7
Total Computation	121582	456	266.6

# CHAPTER 5

## MULTI-GPU PARALLELIZATION OF MLFMA

To speed up the computation of the MoM, the fast multipole method (FMM) has gained widespread use in solving a variety of electromagnetic problems. The FMM was originally proposed by Rokhlin to quickly evaluate particle interactions and to rapidly solve static integral equations [27, 28]. It was extended to solve acoustic wave scattering problems, and then to solve 2D and 3D electromagnetic scattering problems [29, 30]. The FMM reduces the storage and computational complexities from  $O(N^2)$  to  $O(N^{1.5})$ . In order to further speed up the computation and reduce the memory requirement, the recursive variant, the multilevel fast multipole algorithm (MLFMA) was proposed and implemented with  $O(N \log N)$  complexity [24]. In this chapter, a multi-GPU implementation of MLFMA is proposed to take the advantage of GPU advancement. The implementation of the OpenMP-CUDA-MLFMA contains two main parts. One is the calculation of the near-field system matrix  $\mathbf{Z}_{\text{near}}$ . The other is the evaluation of the far-field interaction  $\mathbf{Z}_{\text{far}} \mathbf{I}$ , which includes aggregation, translation and disaggregation phases. In this chapter, we first discuss the multi-GPU implementation of the near-field system matrix assembly. Then we present the parallel strategy for the calculation of the far-field interaction. Finally, we describe the multi-GPU implementation using pinned memory strategy.

### 5.1 Near-Field System Matrix Assembly

In the MLFMA, the computational intensive parts before the iterative solution include the calculation of radiation patterns of the basis functions  $\mathbf{V}_s$  and receiving patterns of the testing functions  $\mathbf{V}_t$ , the calculation of translator  $\mathbf{T}$ , and the assembly of the near-field system matrix  $\mathbf{Z}_{\text{near}}$ . In this section, we mainly discuss the parallelization scheme for the assembly of the near-field

system matrix.

Since the order of basis indices in each group at the finest level is sorted so that the indices of basis functions in each group are continuous, the near-field system matrix  $\mathbf{Z}_{\text{near}}$  has the pattern shown in Figure 5.1(a). This block matrix can be separated into two block matrices. The solid ones in blue represent a block diagonal matrix which comes from self-group interactions. The ones marked by red dash lines represent a block off-diagonal matrix which comes from neighboring-group interactions.

The block matrices are stored as COO format (coordinates list) which is shown in Figure 5.1(b). The nonzero elements are stored in the array  $\mathbf{A}$ . The testing stream  $\mathbf{I}\mathbf{A}$  and the basis stream  $\mathbf{J}\mathbf{A}$  contain the information of the testing and basis functions respectively. To implement the matrix assembly on GPU, the nonzero array  $\mathbf{A}$  is first separated equally into different devices. A one-dimensional (1D) grid of threads is allocated for each device to compute a portion of nonzero elements. During the execution, each thread first fetches the data from the testing stream and the basis stream in the global memory, then calculates a nonzero element following the standard steps of the MoM, and stores the value back to the global memory. The independence of the nonzero elements ensures the efficiency of the hierarchical parallelization regardless of the geometrical shape of the object.

## 5.2 Parallelization on Far-Field Interaction

The parallelization strategy on far-field interaction  $\mathbf{Z}_{\text{far}}\mathbf{I}$  can be implemented by parallelly computing the radiation patterns and receiving patterns of the groups, denoted as  $\mathbf{S}$  and  $\mathbf{B}$ , in the aggregation, translation and disaggregation phases. The basic idea is “one thread per spectrum sampling” and “one/several block(s) per group.” The hierarchical parallelization by simultaneously partitioning groups and their FFPs ensures a high computational throughput for the GPU calculation.

To be specific, take the aggregation phase, for example. The 2D grids and blocks are allocated for the calculation at each level. The block size is set as the size of the spectrum at the finest level. If the mode number is a function of  $kd$ , where  $d$  denotes the maximum diameter of a group and  $k$  is the wavenumber, the spectrum size at the  $(L-1)$ th level (finest level) is  $6 \times 12$

if the length of the cubes at the finest level is set to  $0.3\lambda$ . Figure 5.2 shows the thread allocation at the  $(L-2)$ th level in the aggregation phase. The size of the spectrum at this level is  $12 \times 24$ , where 12 and 24 are the numbers of spectrum in the  $\theta$  and  $\varphi$  direction respectively. Thus four blocks should be organized to represent one parent cube, in which one thread is corresponding to one spectrum. At the  $(L-3)$ th level the spectrum size is  $24 \times 48$ , then 16 blocks are assigned to represent one parent cube and so on. In this way, there will be a sufficient number of threads allocated at each level for parallel computation, which leads to a high computational efficiency.

There are two strategies to implement this parallel idea. One is called global memory strategy; the other is called pinned memory strategy. The global memory strategy requires the radiation and receiving patterns to be calculated and stored at all levels on a single GPU. Such a strategy avoids data transfer between the host and device during aggregation, leading to a very high computational efficiency. However, the size of the global memory will limit the size of problems that can be solved. The pinned memory strategy calculates the radiation and receiving patterns on multiple GPUs, and stores the results to the pinned memory on the host. The benefit of using pinned memory is that we can solve larger problems because the size of the pinned memory is much larger than the global memory. But the data communications between the host and device is unavoidable. To show the capability of solving large problems, the pinned memory strategy is presented and discussed in detail.

### 5.3 Multi-GPU Implementation Using Pinned Memory

Consider  $\mathbf{S}_{i-1}$  at level  $i-1$  aggregated from  $\mathbf{S}_i$  at level  $i$  as shown in Figure 5.3(a). The  $\mathbf{S}_i$  stands for the array lined up with all the groups' radiation patterns at level  $i$ . To facilitate the computation,  $\mathbf{S}_{i-1}$  is equally partitioned into different devices by the number of the groups. The thread allocation in each device is determined by the size of spectrum at level  $i-1$  and the number of parent groups stored in the device. Each device accesses the data from  $\mathbf{S}_i$  stored in the pinned memory, and calculates a part of  $\mathbf{S}_{i-1}$ . Then the results from all the devices are stored back to the pinned memory consecutively. Each matrix-vector product using pinned memory has the

implicit data transfer between the host and device.

The multi-GPU implementation in the translation phase is similar to that in the aggregation phase. Consider  $\mathbf{S}_i$  translating to  $\mathbf{B}_i$  at the same level  $i$  as shown in Figure 5.3(b). The  $\mathbf{B}_i$  stands for the array which consists of all the groups' receiving patterns at level  $i$ .  $\mathbf{S}_i$  is equally divided into different devices by the number of groups. The thread allocation in each device is determined by the size of spectrum at level  $i$  and the number of groups stored in the device. Similarly, each device accesses the data from  $\mathbf{S}_i$  stored in the pinned memory, and calculates a portion of  $\mathbf{B}_i$ . Then the results are stored back to the pinned memory for the use of disaggregation.

The disaggregation phase of MLFMA is very similar to the aggregation phase. As shown in Figure 5.3(c), the partition strategy, thread allocation rule and data communication process are all similar to the ones in the aggregation phase.

## 5.4 Numerical Analysis

In this section, a variety of numerical examples are presented to demonstrate the accuracy and efficiency of the OpenMP-CUDA-MLFMA. The CRWG [22, 23] functions are used as the basis and testing functions to discretize the CFIE ( $\alpha = 0.5$ ). All the numerical examples are solved by the BiCGStab method with a targeted relative residual error of  $10^{-3}$ . The single-precision floating-point arithmetic is used. The CPU-MLFMA and the OpenMP-CUDA-MLFMA are executed respectively on a single CPU (Intel Xeon W3520) using one thread and a 4-GPU system equipped with 4 Nvidia Tesla C2050 GPUs.

### 5.4.1 Validation of the OpenMP-CUDA-MLFMA

#### *Example A: Scattering by a Cone-Sphere with a Gap*

A benchmark model, which is a metallic cone-sphere with a gap at the joint, is simulated to validate the OpenMP-CUDA-MLFMA. The object is 0.689-m long, oriented in the  $z$ -direction, and illuminated by a 3-GHz incident wave. Its surface is discretized into 5006 curvilinear triangular patches with

7509 unknowns. The HH-polarized monostatic radar cross section (RCS) in the  $xz$ -plane are computed, and as can be seen in Figure 5.4(a), a good agreement between the CPU-MLFMA, the OpenMP-CUDA-MLFMA, and the measured data is achieved. Figure 5.4(b) shows the real part of the current density induced on the surface of the scatterer. The variation of the current density can easily be observed.

*Example B: Scattering by a NASA Almond*

The next testing benchmark object is a NASA almond with a size of 25.24 cm  $\times$  9.75 cm  $\times$  3.25 cm. Illuminated by a 9-GHz incident wave, the almond is discretized into 11134 curvilinear triangular patches, resulting in 16701 unknowns. Figure 5.5(a) shows the HH-polarized monostatic RCS in the  $xy$ -plane calculated by the CPU-MLFMA and the OpenMP-CUDA-MLFMA. The measured data are used as reference. Both results agree well with the measured data. The real part of the current density induced by the incident wave is shown in Figure 5.5(b). Through the investigation of the benchmark problems, it is obvious that the accuracy of the OpenMP-CUDA-MLFMA is guaranteed.

In both the cone-sphere and almond examples, the OpenMP-CUDA-MLFMA and CPU-MLFMA results are nearly identical to each other, which indicates that there is no loss of accuracy in the GPU computation.

## 5.4.2 Capability of the OpenMP-CUDA-MLFMA

*Example A: Scattering by a Missile-like Object*

First we consider the electromagnetic scattering of a missile-like object which has a 3-m-long body and 1-cm-thick wings. The nonuniform mesh is employed to discretize the object into 228158 curvilinear triangular patches, leading to 342237 unknowns. Figure 5.6(a) shows the HH-polarized bistatic RCS in the  $xz$ -plane, which demonstrates a good agreement between the results from the OpenMP-CUDA-MLFMA and the CPU-MLFMA. The real part of the current density induced on the surface of the missile-like object is shown in Figure 5.6(b), in which the wave phenomenon can be observed

clearly. The speedups of the OpenMP-CUDA-MLFMA are summarized in Tables 5.1 and 5.2. The same parallelization scheme is applied to the calculation of  $\mathbf{V}_s$  and  $\mathbf{V}_f$ , the translator factor  $\mathbf{T}$ , and the assembly of  $\mathbf{Z}_{\text{near}}$ , which leads to the same speedups for the pinned memory and global memory cases. For the acceleration of the near-field system matrix assembly, a 180 times speedup is achieved. The BiCGstab solver is parallelized using the pinned memory and the global memory strategies respectively, which leads to 75.0 and 19.4 times speedup for the corresponding strategies. The global memory strategy is faster than the pinned memory strategy because there are no data communications between the host and device when calculating the far-field interaction. However, the global memory has the limited size on GPU so that it cannot solve larger problems. Therefore, our discussion is based on the pinned memory strategy in the following larger examples.

#### *Example B: Scattering by Conducting Spheres*

In order to demonstrate the capability and efficiency of the OpenMP-CUDA-MLFMA, the scattering from PEC spheres with diameters of  $4\lambda$ ,  $6\lambda$ ,  $12\lambda$ ,  $30\lambda$  are calculated. The multi-GPU acceleration of the different parts in the MLFMA are investigated as shown in Figure 5.7(a). As can be seen, the excellent speedup is achieved in the near-field system matrix assembly, which increases as the number of unknowns grows. For the different numbers of unknowns, the acceleration in the BiCGstab solution remains the same because the data communications between the host and device take the majority of the time, which is determined by PCIe and front-side bus speed. The total speedup decreases a little bit as the number of unknowns increases, because the BiCGstab solver will take larger portion of the total time. The total speedup achieved is between 30 and 50 times. The HH-polarized bistatic RCS for the  $30\lambda$  sphere with over 1 million unknowns is shown in Figure 5.7(b). The results calculated by the CPU-MLFMA and the OpenMP-CUDA-MLFMA agree well with the Mie series solution. The detailed speedup for the  $30\lambda$  sphere is presented in Table 5.3. The speedup for the near-field system matrix assembly is over 160 times, which is significant. The acceleration of the BiCGstab iterative solution is about 14 times, which is restricted by the data communications between the host and device.

### *Example C: Scattering by an Aircraft*

To further illustrate the capability and efficiency of the proposed method, a more realistic target, which is an aircraft, is considered. The aircraft, with a length of 12.74 m, a width of 15.06 m, and a height of 2.95 m, is illuminated by plane waves with frequencies of 200 MHz, 400 MHz, 780 MHz and 1.5 GHz respectively. The speedup versus number of unknowns is shown in Figure 5.8(a). In the figure, the similar speedups with those for the PEC sphere can be observed. The total speedup is around 20 times. Figure 5.8(b) shows the VV-polarized bistatic RCS calculated by the CPU-MLFMA and the OpenMP-CUDA-MLFMA respectively. The results are on top of each other. The real part of the current density induced on the surface of the aircraft is shown in Figure 5.9 from three different view angles. It is easy to observe the current density variation on the surface of the aircraft. Table 5.4 gives the detailed speedup performance for the aircraft at 1.5 GHz. The speedup for the near-field system matrix assembly is over 160 times, and the speedup of the BiCGstab iterative solution is about 13 times. Comparing Table 5.4 with Table 5.3, it is easy to notice that for the problems with similar number of unknowns, the speedup for each part is similar. In other words, the parallelization scheme is insensitive to the geometrical shape of the object.

## 5.5 Summary

In this chapter, an OpenMP-CUDA based implementation of the MLFMA is presented for computing wave scattering problems of 3D conducting objects on GPU computing systems. For parallelization on a single GPU, a hierarchical parallelization scheme is used by partitioning groups and their FFPs simultaneously. For multi-GPU implementation, a hybrid OpenMP-CUDA parallel programming model is employed. The OpenMP-CUDA-MLFMA is first validated by calculating the monostatic RCS for several benchmark problems. Larger problems are then solved to demonstrate the capability and efficiency of the proposed algorithm. The near-field system matrix assembly using multi-GPU has an excellent efficiency, which has a speedup independent of the object geometry. For the parallelization of the far-field interaction, by the analysis of the GPU architecture and the numerical results, it is revealed

that the global memory strategy is suitable for the fast solution of small problems, and the pinned memory strategy can be employed effectively to accelerate the computation of large problems. The total speedup of the OpenMP-CUDA-MLFMA achieved is between 20 and 80 times, which can be quite important for practical applications.

## 5.6 Figures

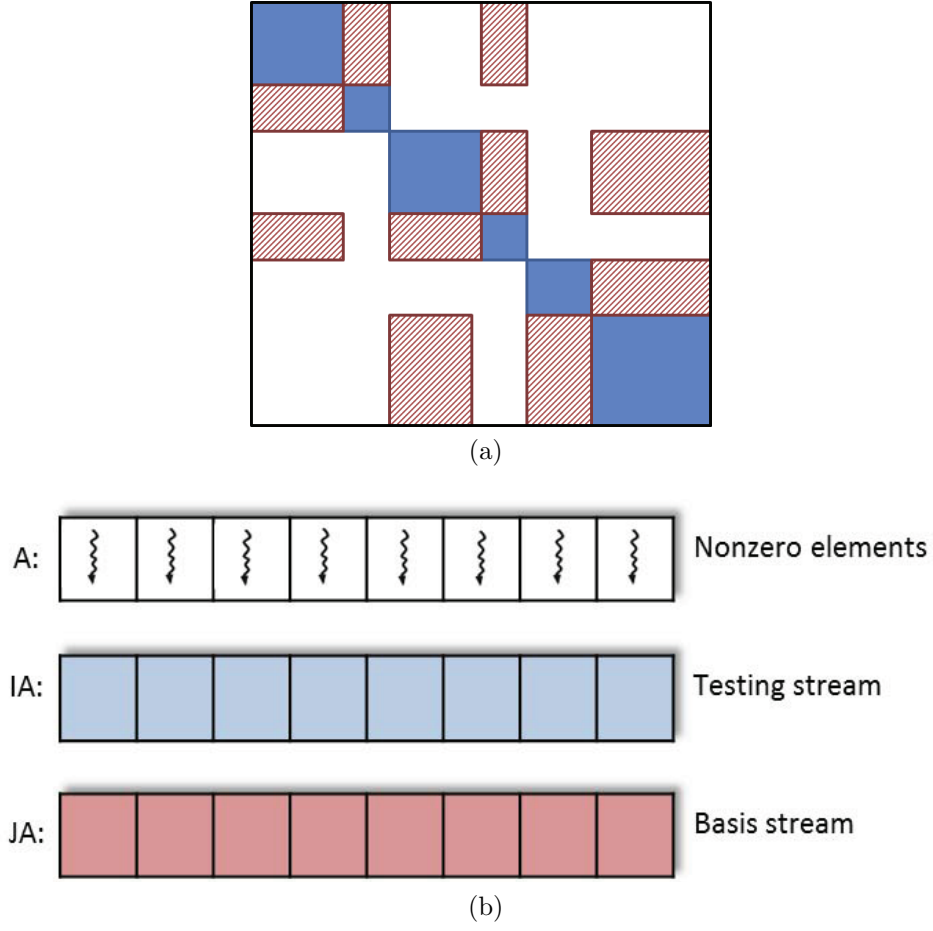


Figure 5.1: Near-field system matrix assembly. (a) Pattern of near-field system matrix. (b) Process of matrix filling.

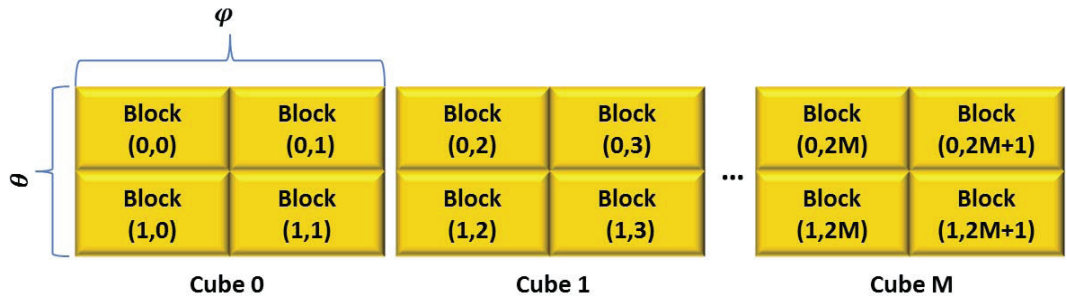
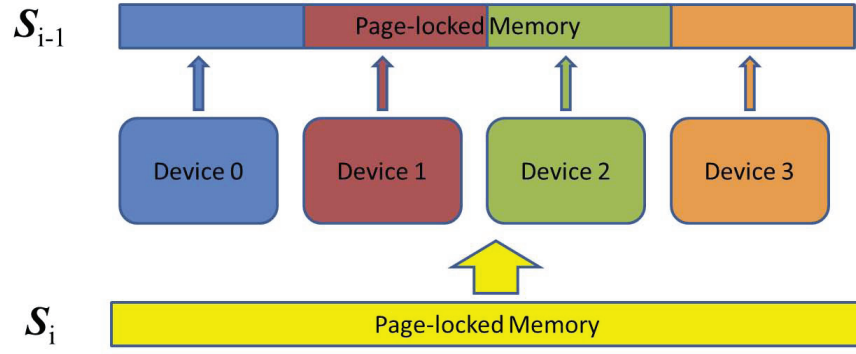
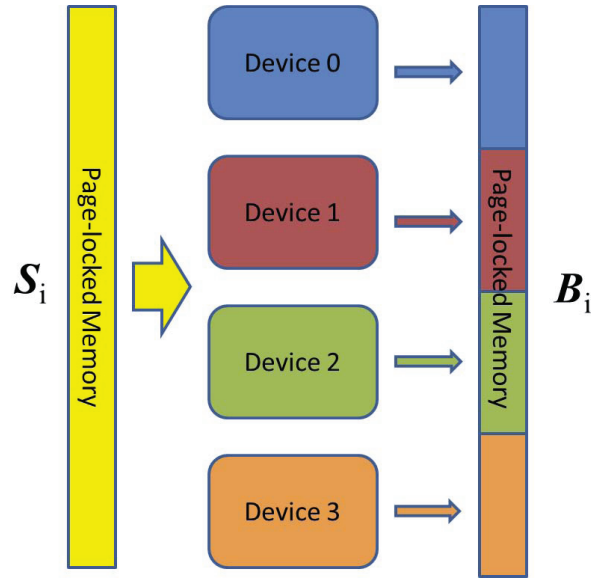


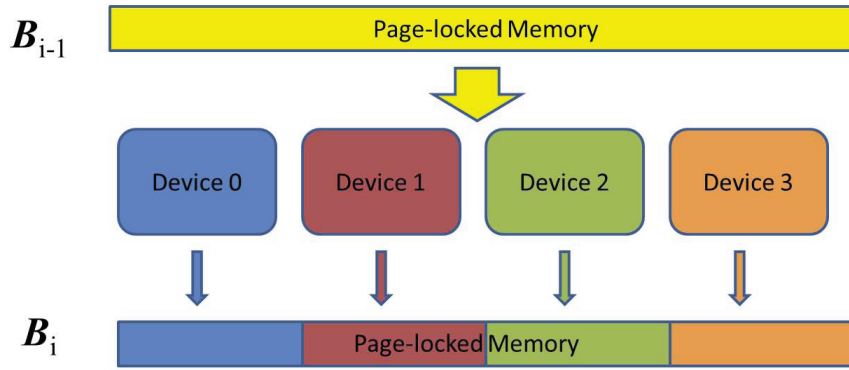
Figure 5.2: Thread allocation for the aggregation phase at the  $(L-2)$ th level.



(a)



(b)



(c)

Figure 5.3: Implementation of far-field interaction on Multi-GPU. (a) Parallel scheme for aggregation. (b) Parallel scheme for translation. (c) Parallel scheme for disaggregation.

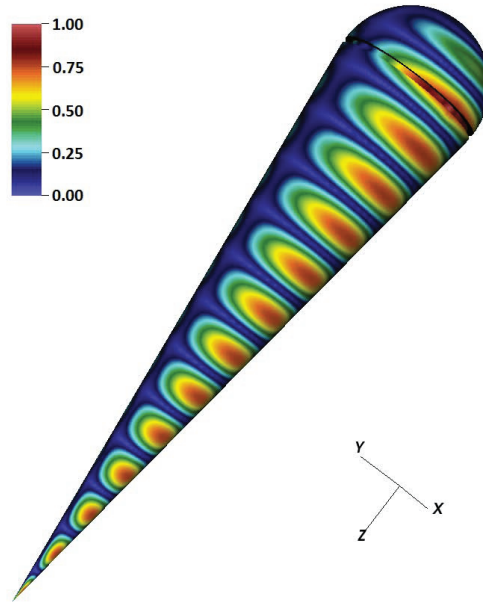
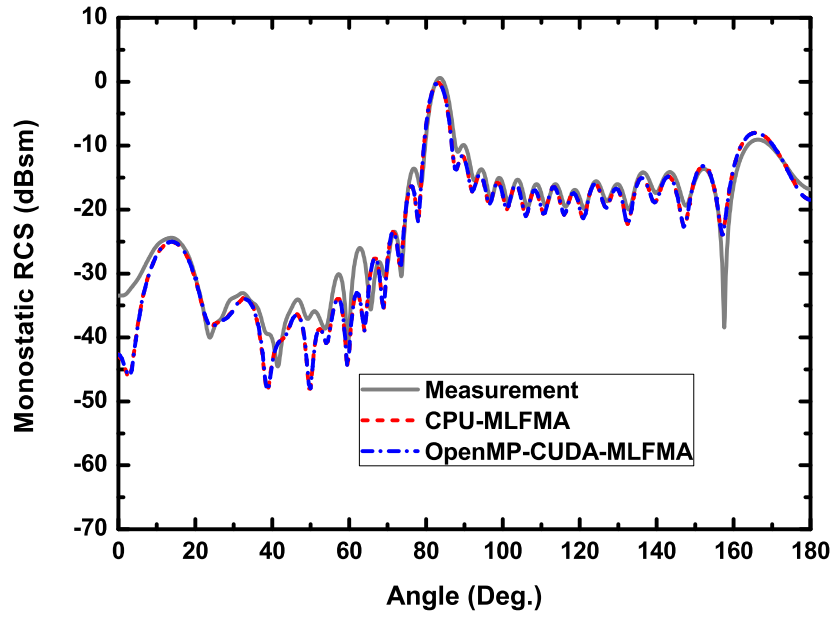


Figure 5.4: Scattering analysis of a cone-sphere with a gap at 3 GHz. The total length of this object is 0.689 m. (a) The HH-polarized monostatic RCS in the  $xz$ -plane. (b) Real part of the current density with the incidence angle  $\theta = 0^\circ$  and  $\phi = 0^\circ$  (in linear scale).

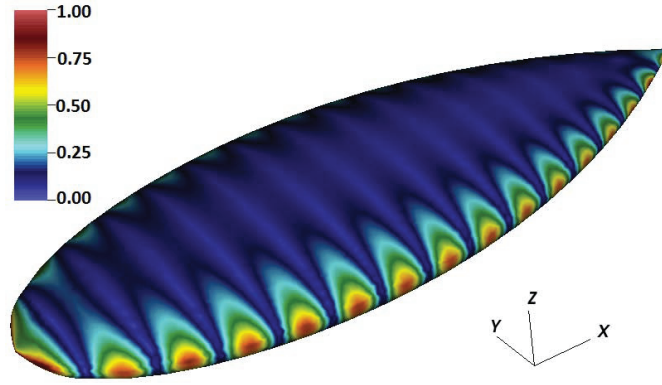
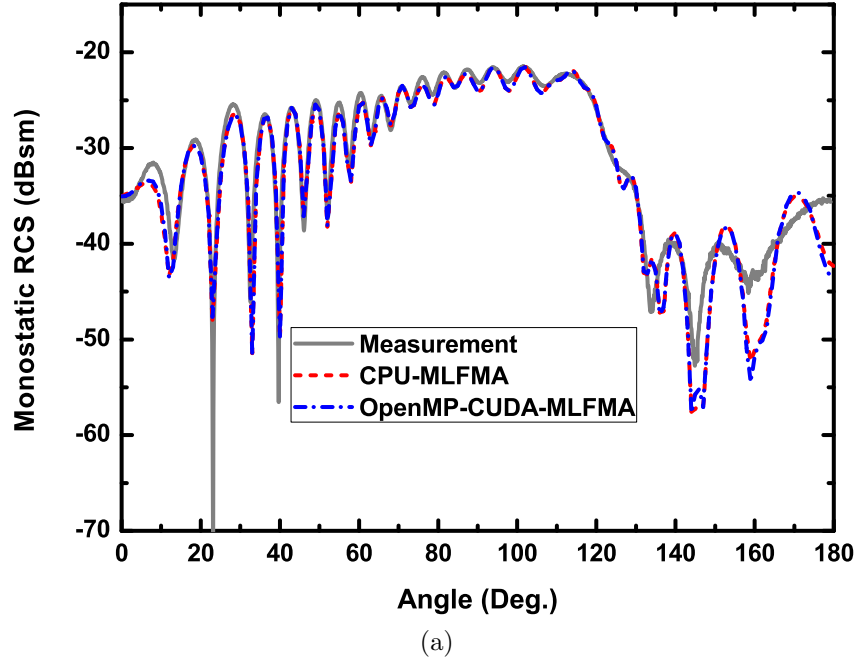
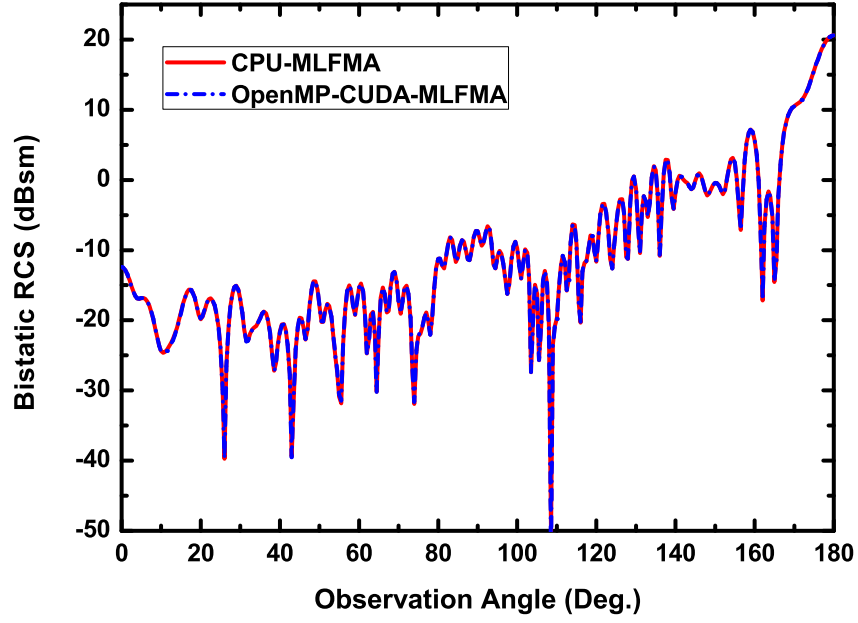
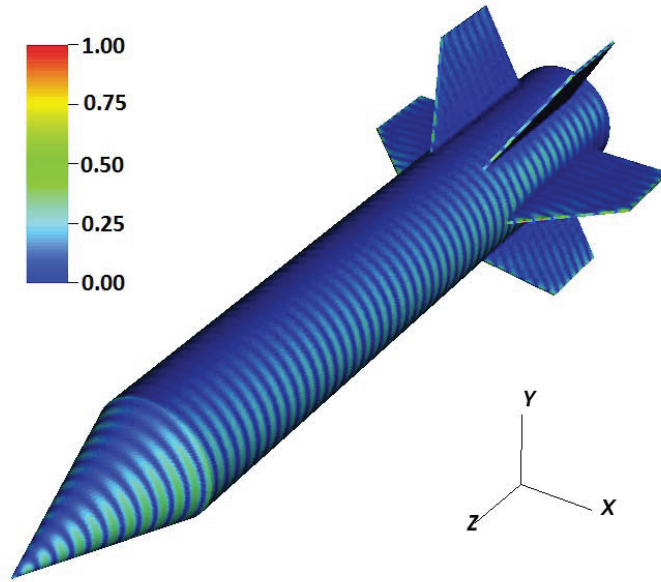


Figure 5.5: Scattering analysis of a NASA almond at 9 GHz. The size of this object is  $25.24 \text{ cm} \times 9.75 \text{ cm} \times 3.25 \text{ cm}$ . (a) The HH-polarized monostatic RCS in the  $xy$ -plane. (b) Real part of the current density with the incidence angle  $\theta = 90^\circ$  and  $\phi = 180^\circ$  (in linear scale).

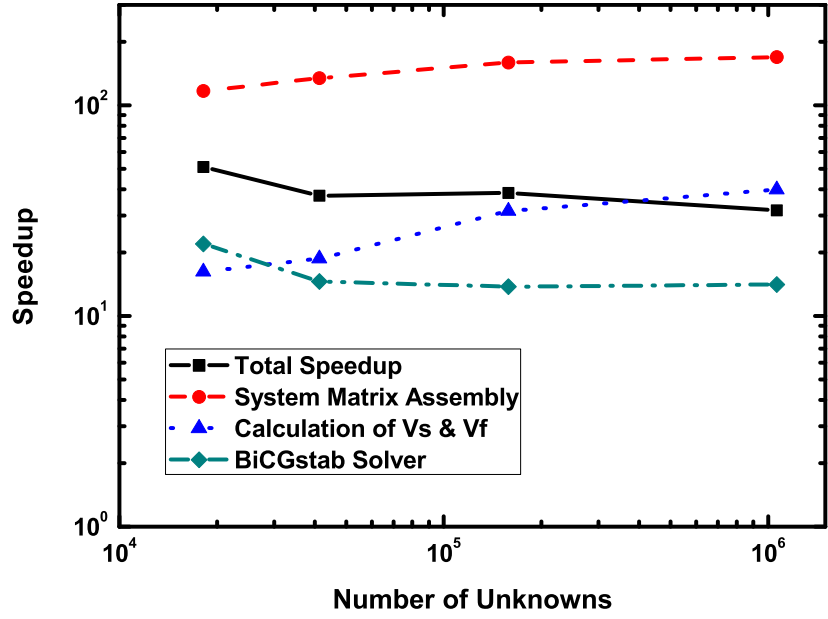


(a)

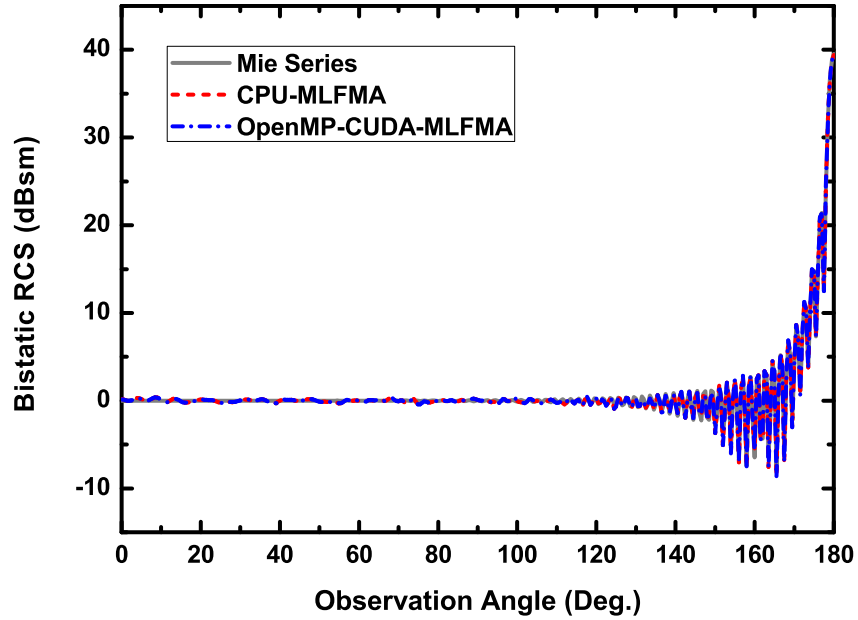


(b)

Figure 5.6: Scattering analysis of a missile-like object. The length of the body is 3 m, and the thickness of the wing is 1 cm. A 3 GHz plane wave is incident from the angle  $\theta = 0^\circ$  and  $\phi = 0^\circ$ . (a) The HH-polarized bistatic RCS in the  $xz$ -plane. (b) Real part of the current density induced on the surface of the scatterer (in linear scale).

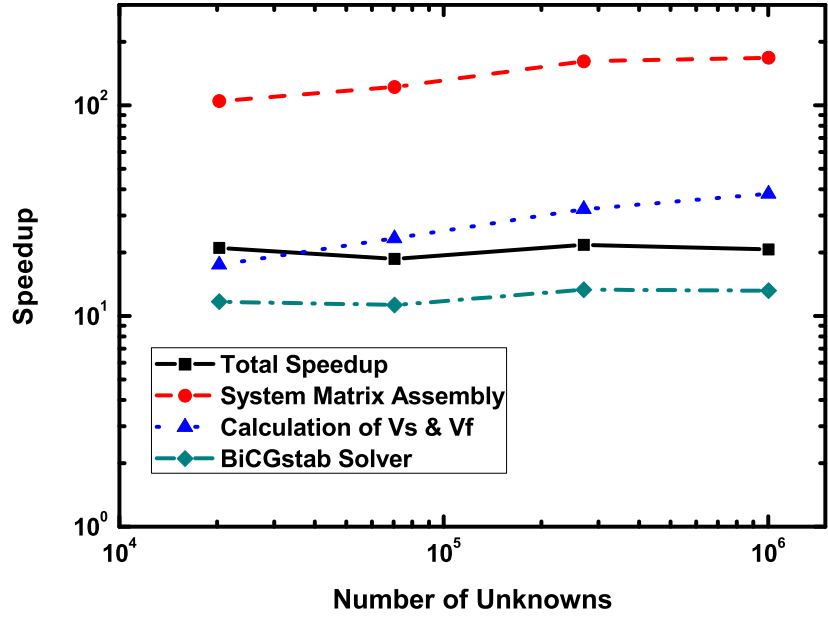


(a)

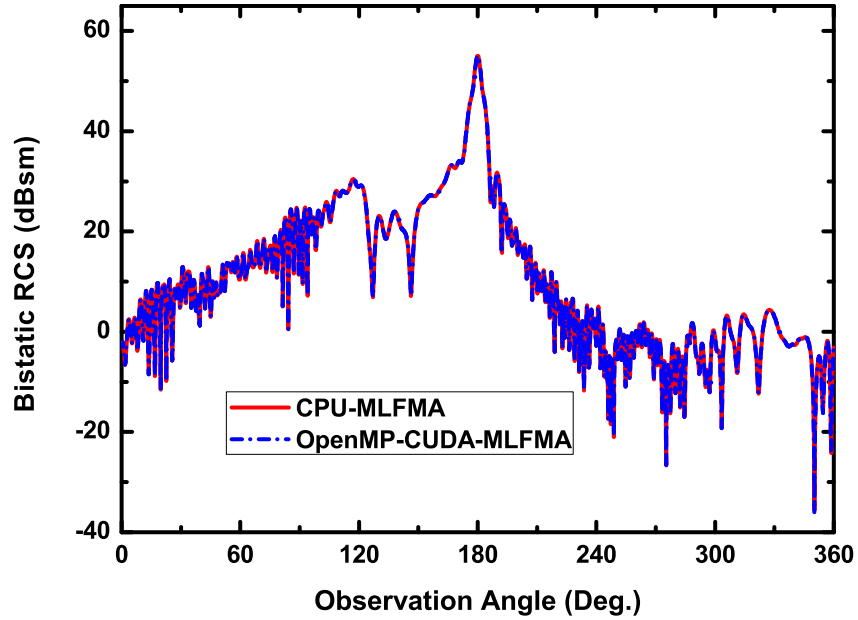


(b)

Figure 5.7: Scattering analysis of the PEC spheres with diameters of  $4\lambda$ ,  $6\lambda$ ,  $12\lambda$ , and  $30\lambda$ . (a) The four devices speedup of the OpenMP-CUDA-MLFMA versus the number of unknowns (unknowns = 18162, 41316, 158333, 1063155). (b) The HH-polarized bistatic RCS of the  $30\lambda$  PEC sphere.



(a)



(b)

Figure 5.8: Scattering analysis of the aircraft at frequencies of 200 MHz, 400 MHz, 780 MHz and 1.5 GHz. (a) The four devices speedup of the OpenMP-CUDA-MLFMA versus the number of unknowns (unknowns = 20319, 70413, 269859, 1001946). (b) The VV-polarized bistatic RCS in the  $yz$ -plane at 1.5 GHz.

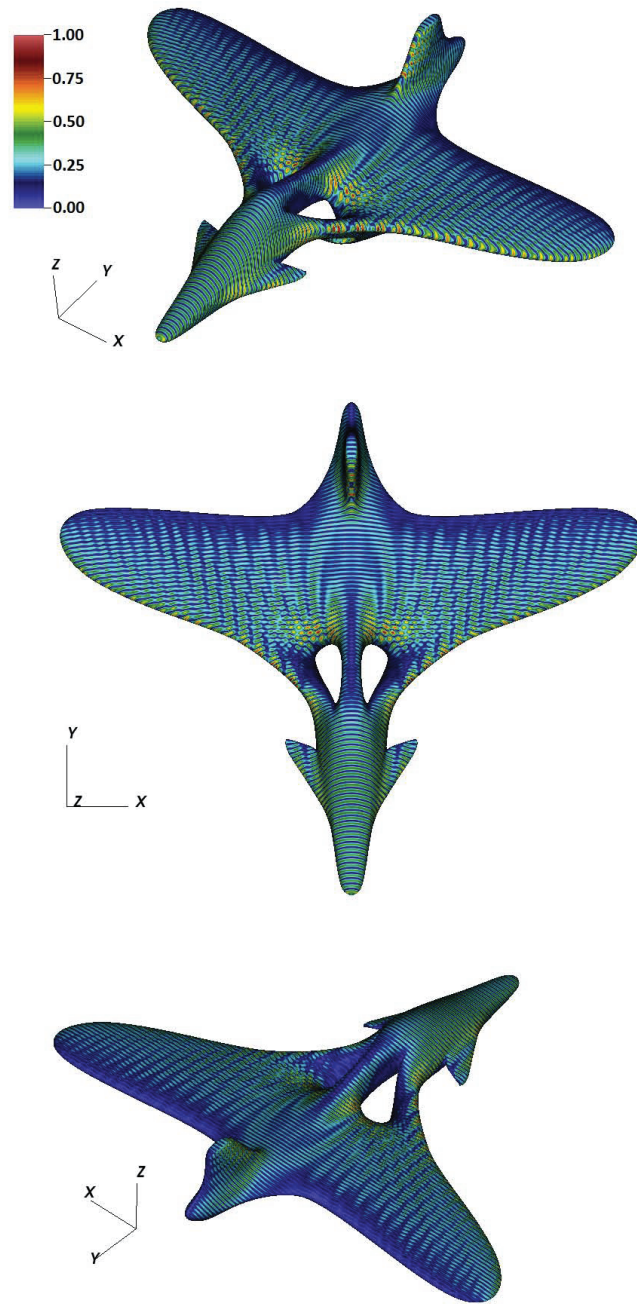


Figure 5.9: Real part of the current density at 1.5 GHz with the incidence angle  $\theta = 60^\circ$  and  $\phi = 270^\circ$  (in linear scale).

## 5.7 Tables

Table 5.1: Speedup of the global memory strategy for bistatic RCS calculation of a missile-like object at 3 GHz

	CPU-MLFMA (sec.)	OpenMP- CUDA-MLFMA (sec.)	Speedup
Calculation of $\mathbf{V}_s$ & $\mathbf{V}_f$	243	7	34.7
Calculation of $\mathbf{T}$	19	1	19.0
Assembly of $\mathbf{Z}_{\text{near}}$	9946	55	180.8
BiCGstab Solution	35180	469	75.0
Total Computation	45388	539	84.2

Table 5.2: Speedup of the pinned memory strategy for bistatic RCS calculation of a missile-like object at 3 GHz

	CPU-MLFMA (sec.)	OpenMP- CUDA-MLFMA (sec.)	Speedup
Calculation of $\mathbf{V}_s$ & $\mathbf{V}_f$	243	7	34.7
Calculation of $\mathbf{T}$	19	1	19.0
Assembly of $\mathbf{Z}_{\text{near}}$	9946	55	180.8
BiCGstab Solution	35180	1811	19.4
Total Computation	45388	1879	24.2

Table 5.3: Speedup of the bistatic RCS calculation of a PEC sphere with diameter of  $30\lambda$

	CPU-MLFMA (sec.)	OpenMP- CUDA-MLFMA (sec.)	Speedup
Calculation of $\mathbf{V}_s$ & $\mathbf{V}_f$	756	19	39.8
Calculation of $\mathbf{T}$	116	0.5	232.0
Assembly of $\mathbf{Z}_{\text{near}}$	7118	42	169.5
BiCGstab Solution	4710	334	14.1
Total Computation	12700	400	31.7

Table 5.4: Speedup of the bistatic RCS calculation of an aircraft at 1.5 GHz

	CPU-MLFMA (sec.)	OpenMP- CUDA-MLFMA (sec.)	Speedup
Calculation of $\mathbf{V}_s$ & $\mathbf{V}_f$	710	19	37.3
Calculation of $\mathbf{T}$	230	1	230.0
Assembly of $\mathbf{Z}_{\text{near}}$	5047	30	168.2
BiCGstab Solution	8631	653	13.2
Total Computation	14618	703	20.8

# CHAPTER 6

## CONCLUSION

In this thesis, the OpenMP-CUDA based implementations of the MoM and MLFMA are presented for electromagnetic simulation on multi-GPU computing systems.

In order to better describe the multi-GPU parallelizations of MoM and MLFMA, the formulations and implementations of MoM and MLFMA are first reviewed. Then the OpenMP-CUDA parallel programming model and GPU architecture are introduced.

MoM, as a basic algorithm, is first accelerated by the use of multi-GPU. The multi-GPU parallel strategies of system matrix assembly, iterative solution and RCS evaluation are discussed in detail. To demonstrate the accuracy and efficiency of the proposed method, electromagnetic scattering of a PEC sphere and a NASA almond are simulated. The accuracy of the proposed method is confirmed by comparing the numerical results with the Mie series solution or measured data. The parallel efficiency versus the number of devices is investigated through the computation of a PEC sphere. As the number of devices increases, the system matrix assembly and RCS evaluation have parallel efficiencies over 95%, while the efficiency of the iterative solution becomes lower. The parallel efficiency of the total computation is over 87%. The total speedup for the monostatic RCS calculation of a NASA almond by 4 GPUs is between 80 and 260 times.

To further speed up the computation of large scattering problems, the OpenMP-CUDA-MLFMA is proposed. The hierarchical parallelization scheme is employed for the implementation by partitioning groups and their FFPs simultaneously. The multi-GPU implementation is developed by hybridizing OpenMP and CUDA parallel programming models. To validate the proposed algorithm, the monostatic RCS for the benchmark problems is calculated. Further, larger problems are solved to demonstrate the capability and efficiency of the proposed algorithm. The near-field system matrix assembly

using multi-GPU has an excellent efficiency, which has a speedup independent of the object geometry. For the multi-GPU accelerated far-field interaction, the global memory strategy is suitable for the fast solution of small problems, and the pinned memory strategy can be employed effectively to accelerate the computation of large problems. The total speedup of the OpenMP-CUDA-MLFMA achieved is between 20 and 80 times, which can be significant in engineering applications.

## REFERENCES

- [1] *NVIDIA CUDA C Programming Guide*, May 2011.
- [2] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors*. Burlington, MA: Morgan Kaufmann, 2010.
- [3] J.-M. Jin, *Theory and Computational Electromagnetic Fields*. Hoboken, NJ: John Wiley & Sons, 2010.
- [4] W. C. Chew, J.-M. Jin, E. Michielssen, and J. M. Song, Eds., *Fast and Efficient Algorithms in Computational Electromagnetics*. Norwood, MA: Artech House, 2001.
- [5] S. Velamparambil, W. C. Chew, and J. M. Song, “10 million unknowns: Is it that big?” *IEEE Antennas Propag. Mag.*, vol. 45, no. 2, pp. 43–58, 2003.
- [6] S. Velamparambil and W. C. Chew, “Analysis and performance of a distributed memory multilevel fast multipole algorithm,” *IEEE Trans. Antennas Propag.*, vol. 53, no. 8, pp. 2719–2727, 2005.
- [7] X.-M. Pan and X.-Q. Sheng, “A sophisticate parallel MLFMA for scattering by extremely large targets,” *IEEE Antennas Propag. Mag.*, vol. 50, no. 3, pp. 129–138, June 2008.
- [8] Ö. Ergül 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, pp. 1740–1750, 2009.
- [9] 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,” *IEEE Trans. Antennas Propag.*, vol. 60, no. 5, pp. 2571–2574, 2012.
- [10] D. D. Donno, A. Esposito, and L. C. L. Tarricone, “Introduction to GPU computing and CUDA programming: A case study on FDTD,” *IEEE Antennas Propag. Mag.*, vol. 53, no. 3, pp. 116–122, June 2010.
- [11] V. Dang, “An implementation of time domain integral equation solution for finite conducting bodies using GPU,” in *The 28th International*

*Review of Progress in Applied Computational Electromagnetics (ACES 2011)*, Columbus, Ohio, Apr. 2011.

- [12] N. Goedel, T. Warburton, and M. Clemens, “GPU accelerated discontinuous Galerkin FEM for electromagnetic radio frequency problems,” in *2009 IEEE AP-S Int. Symp. and URSI Radio Sci. Mtg.*, Charleston, SC, June 2009.
- [13] S. Peng and Z. Nie, “Acceleration of the method of moments calculations by using graphics processing units,” *IEEE Trans. Antennas Propag.*, vol. 56, no. 7, pp. 2130–2133, July 2008.
- [14] T. Topa, A. Noga, and A. Karwowski, “Adapting MoM with RWG basis functions to GPU technology using CUDA,” *IEEE Antennas Wireless Propag. Lett.*, vol. 10, pp. 480–483, 2011.
- [15] T. Topa, A. Karwowski, and A. Noga, “Using GPU with CUDA to accelerate MoM-based electromagnetic simulation of wire-grid models,” *IEEE Antennas Wireless Propag. Lett.*, vol. 10, pp. 342–345, 2011.
- [16] E. Lezar and D. B. Davidson, “GPU-accelerated method of moments by example: Monostatic scattering,” *IEEE Antennas Propag. Mag.*, vol. 52, no. 6, pp. 120–135, Dec. 2010.
- [17] S. Li and V. Lomakin, “Fast iterative electromagnetic integral equation solvers on GPUs,” in *The 28th International Review of Progress in Applied Computational Electromagnetics (ACES 2011)*, Columbus, Ohio, Apr. 2011.
- [18] Y. Liu, V. Lomakin, and E. Michielssen, “Graphics processing unit-accelerated implementation of the plane wave time domain algorithm,” in *The 28th International Review of Progress in Applied Computational Electromagnetics (ACES 2011)*, Columbus, Ohio, Apr. 2011.
- [19] N. A. Gumerov and R. Duraiswami, “Fast multipole methods on graphics processors,” *J. Computational Physics*, vol. 227, no. 18, pp. 8290–8313, Sep. 2008.
- [20] M. Cwikla, J. Aronsson, and V. Okhmatovski, “Low-frequency MLFMA on graphics processors,” *IEEE Antennas Wireless Propag. Lett.*, vol. 9, pp. 8–11, 2010.
- [21] R. F. Harrington, *Field Computation by Moment Methods*. New York: Macmillan, 1968.
- [22] S. M. Rao, D. R. Wilton, and A. W. Glisson, “Electromagnetic scattering by surfaces of arbitrary shape,” *IEEE Trans. Antennas Propag.*, vol. 30, no. 3, pp. 409–418, May 1982.

- [23] R. D. Graglia, D. R. Wilton, and A. F. Peterson, “Higher order interpolatory vector bases for computational electromagnetics,” *IEEE Trans. Antennas Propag.*, vol. 45, no. 3, pp. 329–342, Mar. 1997.
- [24] J. M. Song, C. C. Lu, and W. C. Chew, “Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects,” *IEEE Trans. Antennas Propag.*, vol. 45, no. 10, pp. 1488–1493, Oct. 1997.
- [25] T. G. Mattson, B. A. Sanders, and B. L. Massingill, *Patterns for Parallel Programming*. Boston, MA: Addison-Wesley, 2004.
- [26] J. Sanders and E. Kandrot, *CUDA by Example*. Boston, MA: Addison-Wesley, 2010.
- [27] V. Rokhlin, “Rapid solution of integral equations of classical potential theory,” *J. Computational Physics*, vol. 60, no. 2, pp. 187–207, Sep. 1985.
- [28] L. Greengard and V. Rokhlin, “A fast algorithm for particle simulations,” *J. Computational Physics*, vol. 73, no. 2, pp. 325–348, Dec. 1987.
- [29] V. Rokhlin, “Rapid solution of integral equation of scattering theory in two dimensions,” *J. Computational Physics*, vol. 86, no. 2, pp. 414–439, Feb. 1990.
- [30] R. Coifman, V. Rokhlin, and S. Wandzura, “The fast multipole method for the wave equation: A pedestrian prescription,” *IEEE Antennas Propag. Mag.*, vol. 35, no. 3, pp. 7–12, June 1993.