THE SOLUTION OF LARGE-SCALE ELECTROMAGNETIC PROBLEMS WITH MLFMA ON SINGLE-GPU SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRIC AND ELECTRONICAL ENGINEERING

By
Mehmet Fatih Erkal
January 2022

The Solution of Large-Scale Electromagnetic Problems with MLFMA on Single-GPU Systems
By Mehmet Fatih Erkal
January 2022

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Vakur Behçet Ertürk(Advisor)

Barışcan Karaosmanoğlu(Co-Advisor)

Ergin Atalar

Hatice Özlem Aydın Çivi

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

THE SOLUTION OF LARGE-SCALE ELECTROMAGNETIC PROBLEMS WITH MLFMA ON SINGLE-GPU SYSTEMS

Mehmet Fatih Erkal
M.S. in Electric and Electronical Engineering
Advisor: Vakur Behçet Ertürk
January 2022

Advancements in computer technology introduce many new hardware infrastructures with high-performance processing powers. In recent years, the graphics processing unit (GPU) has been one of the popular choices that have been being used in computational engineering fields because of its massively parallel processing capacity and its easy coding structure compatible with new programming systems. The full-wave solution of large-scale electromagnetic (EM) scattering problems with traditional methods has very dense computational operations, and thus additional hardware accelerations become an indispensable demand, especially for practical and industrial applications. In this context, the GPU implementation of full-wave electromagnetic solvers such as multi-level fast multiple algorithm (MLFMA) has shown a trend in the literature for the last decade. However, the GPUs also have many restrictions and bottlenecks when implementing large-scale EM scattering problems with full-wave solvers. Limited random-access-memory (RAM) capacities and data transmission delays are the major bottlenecks. In this study, we propose a matrix partitioning scheme to overcome the RAM restriction of GPUs to be able to solve electrically large size problems in single-GPU systems with MLFMA while acquiring reasonable accelerations by considering different implementation approaches. For this purpose, the Single-Instruction-Multiple-Data (SIMD) structure of GPU is considered for each stage of the MLFMA to check its compatibility. In addition, different operators of MLFMA are fine-tuned on the GPU scale to minimize the overall effect of data transfer and device latency. The preliminary analyses show that significant time efficiencies can be obtained for the different parts of MLFMA as well as eliminating the RAM restriction. The numerical results demonstrate the overall efficiencies of our proposed solution for the bottlenecks of GPU and also validate the expected accelerations for the solution of large-scale

EM problems involving electrically large canonical geometries and real-life targets such as an aircraft and a missile geometry.

Keywords: Fast algorithms, MLFMA, hardware acceleration, GPU.

ÖZET

BÜYÜK ÖLÇEKLİ ELEKTROMANYETİK PROBLEMLERİN ÇSHÇY İLE TEKLİ-GİB SİSTEMLERİNDE ÇÖZÜMÜ

Mehmet Fatih Erkal
Elektrik Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Vakur Behçet Ertürk
Ocak 2022

Bilgisayar teknolojisindeki gelişmeler, yüksek performanslı işlem gücüne sahip birçok yeni donanım altyapısını beraberinde getiriyor. Son yıllarda grafik işlem birimi (GİB), yüksek güçteki paralel işlem kapasitesi ve yeni programlama sistemlerine uyumlu kolay kodlama yapısı nedeniyle hesaplamalı mühendislik alanlarında kullanılan popüler seçeneklerden biri olmuştur. Büyük ölçekli elektromanyetik (EM) saçılım problemlerinin geleneksel yöntemlerle tam dalga çözümü, çok yoğun hesaplama işlemlerine sahiptir ve bu nedenle, özellikle pratik ve endüstriyel uygulamalar için ek donanım hızlandırmaları vazgeçilmez bir talep haline gelir. Bu bağlamda, çok seviyeli hızlı çokkutup yöntemi (ÇSHÇY) gibi tam dalga elektromanyetik çözücülerin GİB uygulaması, son on yılda literatürde bir yönelim göstermiştir. Bununla birlikte, tam dalga çözücülerle büyük ölçekli EM saçılım problemlerini GİB'lerde uygulamanın birçok kısıtlama ve zorluğu vardır. Sınırlı rasgele erişimli bellek (REB) kapasiteleri ve veri iletim gecikmeleri başlıca kısıtlamalardır. Bu çalışmada, ÇSHÇY ile tek GİB'li sistemlerde elektriksel olarak büyük boyutlu problemleri çözebilmek ve farklı uygulama yaklaşımlarını dikkate alarak makul hızlanmalar elde edebilmek için GİB'lerin REB kısıtlamasını aşmak için bir matris bölümleme şeması öneriyoruz. Bu amaçla, ÇSHÇY 'nin her aşamasının uyumluluğunu kontrol etmek için GİB'in Tek Komutlu-Çoklu Veri (TKÇV) yapısı dikkate alınır. Ayrıca, veri aktarımının ve cihaz gecikmesinin genel etkisini en aza indirmek için farklı ÇSHÇY operatörleri GİB ölçeğinde hassas olarak ayarlanmıştır. Ön analizler, REB kısıtlamasının ortadan kaldırılmasının yanı sıra, ÇSHÇY 'nin farklı bölümleri için önemli zaman verimlerinin elde edilebileceğini göstermektedir. Sayısal sonuçlar, GİB kısıtlamaları için önerilen çözümümüzün genel verimliliğini gösterir ve ayrıca elektriksel olarak büyük kanonik geometrileri ve bir uçak ve bir füze geometrisi

gibi gerçek hayat hedeflerini içeren büyük ölçekli EM problemlerinin çözümü için beklenen hızlanmaları doğrular.

Anahtar sözcükler: Hızlı algoritmalar, ÇSHÇY, donanım hızlandırması, GİB.

Acknowledgement

Firstly, I would like to express my profound appreciation to my advisor Prof. Vakur Behçet Ertürk for his supervision, guidance, support and patience throughout my master studies at Bilkent University. He always pushed me to be a more successful researcher. I am greatly appreciated all of his invaluable comments and ideas, which pointed me in the right direction in my research studies.

I would like to thank to my co-advisor, Barışcan Karaosmanoğlu, for his supervision, support, and encouragement during the time we have worked together.

I also would like to thank members of examination committee, Prof. Ergin Atalar and Prof. Hatice Özlem Aydın Çivi for accepting to participate in my jury and their valuable comments about my thesis.

I would like to thank all of my friends and colleagues who always give their supports and encouragement to me all the time.

Last but not least, I must express my endless appreciation to my family. They support me and believe me under all circumstances, all the time. I want to express my special thanks to my father, mother, and grandmother.

Contents

1 Introduction **1**

2 Theoretical Background **5**

2.1 Solution Mechanism of EM Scattering Problems 6

2.2 Surface Integral Equations 8

2.2.1 Basics of Electromagnetic Theory 8

2.2.2 Equivalence Principle 13

2.2.3 Formulation of SIEs 15

2.3 Method of Moments 16

2.3.1 RWG Basis Function 17

2.3.2 Galerkin Scheme and Matrix Elements of SIEs 18

2.4 Numerical and Analytical Integration 20

2.4.1 Gaussian Quadrature Technique 20

2.4.2 Singularity Extraction of Integral Equations 21

2.5 Multi-Level Fast Multiple Algorithm 23

2.5.1 Addition Theorem for the Green's Function and Diagonalization 24

2.5.2 Clustering of FMM and One-Box Buffer Scheme 27

2.5.3 Stages of MLFMA 32

2.5.3.1 Aggregation 32

2.5.3.2 Translations 32

2.5.3.3 Disaggregation 33

3 MLFMA Implementation on GPU Systems 34

3.1 Graphics Processing Units 34

3.2 Implementation of MLFMA on GPU 37

3.3 Bottlenecks of GPU and Possible Remedies for the Solution of Large-Scale Problems 42

3.3.1 Bottlenecks of GPU 42

3.3.2 Possible Remedies and Implementation Approaches 43

4 Numerical Results & Discussion 48

4.1 Validation Results 49

4.1.1 Sphere Analyses 49

4.1.2 Cube Analyses 52

4.2 MLFMA GPU Implemention Results 54

4.2.1 Preliminary Analyses for Testing Different GPU Implementation Approaches 54

4.2.2 Sphere Results Solved on PC 64

4.2.3 Sphere Results Solved on Server 74

4.2.4 Aircraft Results 83

4.2.5 Missile Results 88

5 Conclusion and Recommendations 93

A Scattered Fields of Sphere Analyses 103

A.1 Results solved on PC 104

A.2 Results solved on the server 106

B Current Distributions of Sphere Analyses 109

B.1 Results solved on PC 110

B.2 Results solved on the server 112

List of Figures

2.1 Summary of the Solution Mechanism 7

2.2 Test and Source Point Representation 11

2.3 Definition of the actual problem and the equivalent problem . . . 13

2.4 Surface boundary of a PEC object exposed to an electromagnetic radiation 14

2.5 Definition of RWG Function 18

2.6 Group-by-group manner interactions for FMM 23

2.7 Interaction with Multiple Shift 26

2.8 FMM Clustering 28

2.9 Multi-level group-by-group interaction 29

2.10 MLFMA Clustering 30

2.11 MLFMA child-parent relations of boxes 31

2.12 MLFMA child-parent relations of boxes 31

3.1 CPU and GPU cores 35

3.2 GPU architecture overview [55] 36

3.3 MLFMA data structures of different variables 38

3.4 Illustration of MLFMA stages 41

3.5 General scheme of GPU implementation 43

3.6 Cyclic process of the proposed GPU implementation 44

3.7 Partitioning scheme of different data structures of MLFMA 45

4.1 Illumination of a PEC sphere geometry by an x-polarized ([1 0 0]), +z-propagated ([0 0 1]) plane wave. 50

4.2 The comparison of scattered fields (MoM and MLFMA) with the Mie series solution ((a),(b)) and induced currents ((c),(d)) for different electrical sizes of spheres used for validation analyses. . . 51

4.3 Illumination of a PEC cube geometry by an x-polarized ([1 0 0]), +z-propagated ([0 0 1]) plane wave. 52

4.4 The comparison of scattered fields between MoM and MLFMA solutions ((a),(b)) and induced currents ((c),(d)) for different electrical sizes of cubes used for validation analyses. 53

4.5 Near-field timings for the first approach of GPU implementation solved on PC 55

4.6 Near-field average timings for the first approach of GPU implementation solved on PC 56

4.7 A part of the log-file of a GPU implementation for the calculation of near-field matrix solved on PC 57

4.8 Radiation/receiving pattern timings for the first approach of GPU implementation solved on PC 58

4.9 Interpolation & shifting timings for the first approach of GPU implementation solved on PC 59

4.10 Timings of multiplication process of translation subroutine for the first approach of GPU implementation solved on PC 60

4.11 Near-Field timing test for the second approach of GPU implementation solved on PC 61

4.12 Radiation/Receiving pattern timing test for the second approach of GPU implementation solved on PC 62

4.13 The resulting scattered fields compared with Mie series solution for the sphere analyses of $1\lambda, 16\lambda, 48\lambda$ spheres solved on PC . . . 67

4.14 The resulting induced currents for the sphere analyses of $1\lambda, 16\lambda, 48\lambda$ spheres solved on PC 68

4.15 Near-Field and MVM and total times for sphere analyses solved on PC 71

4.16 Details of MVM timing values for sphere analyses solved on PC . 72

4.17 The resulting scattered fields compared with Mie series solution for the sphere analyses of $8\lambda, 32\lambda, 56\lambda, 80\lambda$ spheres solved on the server 77

4.18 The resulting induced currents for the sphere analyses of $8\lambda, 32\lambda, 56\lambda, 80\lambda$ spheres solved on the server 78

4.19 Near-Field and MVM and total times for sphere analyses solved on the server 81

4.20 Details of MVM timings for sphere analyses solved on server . . . 82

4.21 Aircraft Geometry 83

4.22 Illumination of the aircraft model with horizontal and vertical polarizations 84

4.23 Induced currents on the aircraft geometry 84

4.24 The comparison of scattered fields from the aircraft geometry by the reference [16] 85

4.25 Scattered E-field of the aircraft geometry when illuminated by an H-pol plane wave 86

4.26 Scattered E-field of the aircraft geometry when illuminated by a V-pol plane wave 86

4.27 Missile geometry 88

4.28 Illumination of the missile model with horizontal and vertical polarizations 89

4.29 Induced currents on missile geometry 89

4.30 The comparison of scattered fields from the missile geometry with an independent reference 90

4.31 Scattered E-field from missile model polar plot 91

4.32 Scattered E-field from missile model polar plot 91

A.1 The resulting scattered fields compared with MIE series solution for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on PC 104

A.2 The resulting scattered fields compared with MIE series solution for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda$ spheres solved on PC 105

A.3 The resulting scattered fields compared with MIE series solution for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on the server 106

A.4 The resulting scattered fields compared with MIE series solution for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda, 56\lambda$ spheres solved on the server 107

A.5 The resulting scattered fields compared with MIE series solution for the sphere analyses of $64\lambda, 72\lambda, 80\lambda, 88\lambda$ spheres solved on the server 108

B.1 The resulting induced currents for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on PC 110

B.2 The resulting induced currents for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda$ spheres solved on PC 111

B.3 The resulting induced currents for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on the server 112

B.4 The resulting induced currents for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda, 56\lambda$ spheres solved on the server 113

B.5 The resulting induced currents for the sphere analyses of $64\lambda, 72\lambda, 80\lambda, 88\lambda$ spheres solved on the server 114

List of Tables

4.1 Specifications of sphere problems solved on PC 66

4.2 Time comparison of CPU and GPU implementations for different sphere geometries solved on PC. 69

4.3 Specifications of sphere problems solved on the server 76

4.4 Time comparison of CPU and GPU for different sphere geometries solved on the server 79

4.5 Time comparison of CPU and GPU for different sphere geometries solved on the server 80

4.6 Time comparison of CPU and GPU implementations for the aircraft geometry illuminated by the H-pol plane-wave 87

4.7 Time comparison of CPU and GPU implementations for the aircraft geometry illuminated by the V-pol plane-wave 87

4.8 Time comparison CPU vs. GPU for missile geometry illuminated by H-pol 92

4.9 Time comparison CPU vs. GPU for missile geometry illuminated by V-pol 92

List of Abbreviations

ADI-PE:	Alternating direction implicit-parabolic equation
CAD:	Computer-aided design
CEM:	Computational electromagnetics
CFIE:	Combined-field integral equation
CPU:	Central processing unit
EFIE:	Electric-field integral equation
EM:	Electromagnetics
FE-BI:	Finite element-boundary integral
FMM:	Fast multipole method
GMRES:	Generalized minimal residual (method)
GPGPU:	General-purpose computation on GPU
GPU:	Graphics processing unit
HPC:	High-performance computing
IE:	Integral equation
MFIE:	Magnetic-field integral equation
MLFMA:	Multilevel fast multipole algorithm
MoM:	Method of moments
MVM:	Matrix-vector multiplication

PC:	Personal computer
PEC:	Perfect electric conductor
PCI-E:	Peripheral component interconnect-express
PW:	Plane wave
RAM:	Random access memory
RCS:	Radar cross-section
RWG:	Rao-Wilton-Glisson (function)
SIE:	Surface integral equation
SIMD:	Single instruction multiple data
TPB:	Thread-per-box
TPS:	Thread-per-sample
VIE:	Volume integral equation

Chapter 1

Introduction

Studies about electromagnetic (EM) radiation and scattering problems have gained particular importance for the last past decades as antenna and radar systems have become a substantial and indispensable part of modern technological systems. Especially in the defense industry, the significance of calculating the radar cross-section (RCS) or verifying the antenna radiation patterns is critical. The design of advanced defense systems like aircraft or ships has very complex and sinuous geometries. So, analytical calculations of electromagnetic scattering for these types of geometries are cumbersome. Thus, numerical approaches are necessary. On the other hand, real-life EM scattering problems are electrically large as the wavelength of the frequency is small concerning the geometry's physical dimensions. In addition, many iterations are required during the design processes for healthy verification. Therefore, efficient solution methods and sufficient computational power are required to solve the large scattering EM problems in reasonable times.

Maxwell equations have been proven as the governing equations of electromagnetic problems [1, 2]. Full-wave numerical methods such as integral-equation (IE) methods can be used to discretize Maxwell's equations on the surface of homogeneous objects while satisfying the radiation boundary conditions using equivalent sources on the surface of the objects [3–5]. Since the equivalent

sources are defined only on the surface of the object, the region of interest in the entire EM problem reduces to surface part, and integral equations are named as surface integral equations (SIEs). Method of moments (MoM) is one of the well-known mathematical methods to solve electromagnetic scattering problems by using integral equations [2, 6–8]. However, because of the high computational complexity ($O(N^2)$) of MoM [2], fast and efficient methods have been developed, such as Fast Multiple Method (FMM) and its multi-level version Multi-Level Fast Multiple Algorithm (MLFMA) to reduce the computational complexity into $O(N^{3/2})$ and $O(Nlog(N))$, respectively [3, 4, 9–12]. In addition, these efficient algorithms can be further accelerated via machine learning based implementations [13–16]. Note that, both in FMM and MLFMA methods, the error is controlled [17–19].

These techniques, FMM/MLFMA, constitute one side of the computational analysis. The other side is computational power. Advancements in computer technology offer us many new high-power hardware infrastructures such as High-Performance-Computing (HPC) systems, graphics processing units (GPUs) with many cores and large RAMs, servers, etc. So, applying fast and efficient solution techniques (FMM, MLFMA) on high-performance hardware architectures (such as GPU) can enable us to accelerate the solution of large-scale electromagnetic problems. The hardware infrastructure of GPUs has been developed rapidly in recent years [20], and GPUs have become a good alternative for the efficient solutions of computational engineering problems with its massive parallel process power [21–23].

Using GPU architectures for the solution of electromagnetic problems has become popular in the recent years [24–26]. We investigate different publications to see which methods/techniques are performed and how much time efficiency is obtained for the GPU implementations. Speed-up comparisons between different studies cannot be made since the specifications of GPU cards are different. However, an idea of the reported speed-up values and implementation methods/techniques can help us to enhance our investigation strategies.

In the literature, there are many publications about various electromagnetic problems implemented on GPU. For different integral equations (SIE [27–30], FE-BI [31, 32], ADI-PE [33]), different GPU implementations have been conducted on different GPU hardware infrastructures (single-GPU [29,30], multi-GPU [27] and GPU clusters [34, 35]). These implementations use different mathematical approaches as MoM [24, 36, 37], FMM [25] or MLFMA [26–32, 34, 35]. There are different kind of techniques to utilize the GPU for the MoM and FMM/MLFMA algorithms. Thread-per-sample (TPS)/thread-per-direction approach and thread-per-box (TPB)/thread-per-subdomain approach are the common techniques used in the literature for MoM [24, 36, 37], FMM [25] and MLFMA [29,30,34,35]. Most of the papers report that single-precision arithmetic is used [27, 29, 30] and a few of them use double-precision [24, 28]. Nevertheless, only a few of them exhibits CFIE formulation with MLFMA approach on single-GPU architectures [29, 30].

This thesis focuses on the efficient solutions of large-scale electromagnetic problems using the SIE formulations with MLFMA on single-GPU systems under double-precision arithmetic. Our objective is to solve the large-scale EM scattering problems on single-GPU cards by eliminating the bottlenecks of GPU (such as RAM (random access memory) capacities, memory access delays, transmission delays, etc. [29, 30, 38, 39]) while acquiring reasonable accelerations. RAM limitation of GPU cards is the biggest restriction to solve electrically large targets in the literature [26, 27, 29, 30]. Therefore, in this study, we propose a matrix partitioning scheme to eliminate the RAM limitation while taking advantage of the high parallel computing power of GPU cards. Since the data sizes (or memory requirement) in MLFMA gets higher as the electrical size increases, one solution can be dividing the data (to be processed) into small partitions that can fit into the GPU RAM and then process the data partitions separately in order, one by one, until the entire calculation is completed. In this scheme, the compatibility of the algorithmic process in the GPU must be a Single-Instruction-Multiple-Data (SIMD); in other words, it must consist of many small independent units of the same work. Furthermore, possible side-effects can be the data transmission and/or device/memory access/initialization delays,

and these delays must be taken into account to obtain a notable acceleration in the overall execution. Otherwise, the effects of transmission/device delays may result in performance degradation instead of acceleration. Thus, for an efficient GPU implementation with our proposed scheme, we analyze the different parts of MLFMA to show which parts are compatible with SIMD and which parts are not. Moreover, different implementation approaches are offered for data partitioning to find an optimum method to obtain the best efficiency in GPU implementation. Regarding these concerns, large-scale EM problems can be solved in single-GPU systems with reasonable accelerations with respect to single-CPU systems. The numerical examples exhibited in this study are formulated with SIE formulations to model the scattering problems from homogeneous perfect electric conductor (PEC) bodies under plane-wave (PW) illumination.

The organization of the rest of this thesis is as follows: In Chapter 2, theoretical background is given from Maxwell's equations to MLFMA and the required derivations. In Chapter 3, GPU architecture is explained, and then the compatible parts of MLFMA with SIMD implementation are investigated. Finally, bottlenecks and possible methods are considered for an efficient GPU implementation. In Chapter 4, numerical results are given. This chapter consists of validation results and MLFMA GPU implementation results. In Chapter 5, the work of this thesis is concluded, and the recommendations for future studies are given. In Appendix A and Appendix B, scattered fields (compared with Mie solutions) and induced currents of sphere analyses done in Chapter 4.2 are given, respectively.

Throughout this work, an $e^{j\omega t}$ time convention is used and suppressed, with an angular frequency of $\omega = 2\pi f$. The generalized minimal residual method (GMRES) [40] is used as the iterative solver for the solution of matrix equations.

Chapter 2

Theoretical Background

In this chapter, firstly the solution mechanism of an EM scattering problem is summarized. Afterwards, the surface integral equations are studied by investigating the basics of electromagnetic theory and equivalence principle. Then, a linear system of equations is obtained in a matrix form by using the method of moments (MoM). Commonly used numerical and analytical integration techniques are studied for the solution of integral equations. Finally, mathematical techniques used with iterative solvers are examined for the efficient solution of matrix equations.

2.1 Solution Mechanism of EM Scattering Problems

The solution mechanism of electromagnetic scattering problems describes the procedure of obtaining a numerical solution from a physical problem. In this stream, when we deal with electromagnetic scattering from impenetrable objects, Maxwell's equations and boundary conditions are used to obtain surface integral equations. Surface integral equations are discretized and tested/weighted by a proper testing/weighting functions which yields a matrix equation (called MoM matrix equation). Gaussian quadrature and/or analytical integration techniques are used to solve the surface integral equations in the matrix form. In this process, iterative solvers are widely used and accelerated with some mathematical techniques such as FMM or its multi-level version MLFMA. Once the solution is obtained, the scattered (or secondary) field is calculated. In order to solve the surface integral equations in the matrix equation form, proper geometrical discretization is of great importance. For this purpose, computer-aided design (CAD) models of the objects need to be simplified and discretized carefully. The entire procedure (see Figure 2.1) can be considered in three main stages, namely theory & formulation, modeling & meshing and solution & post-calculation.

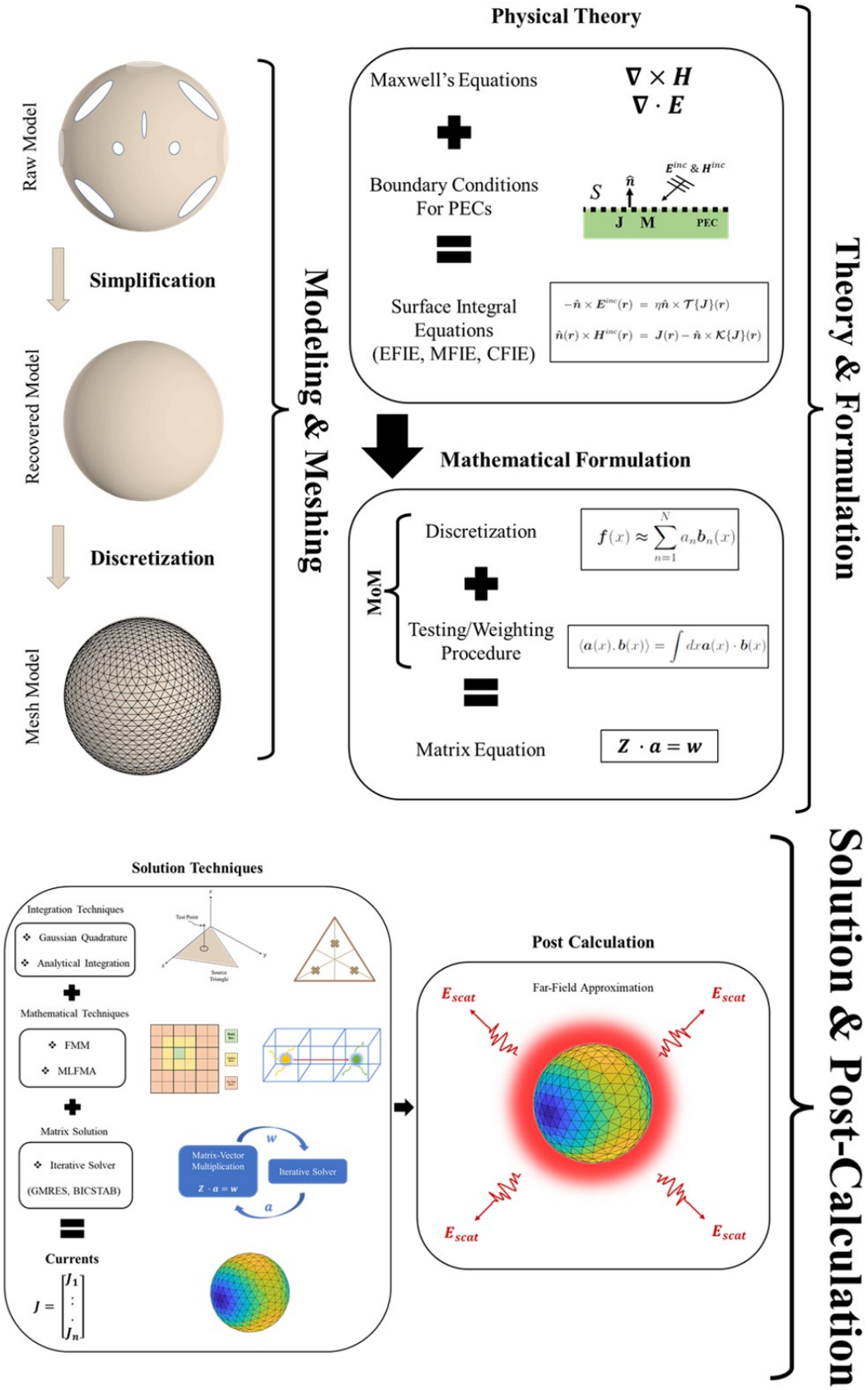


Figure 2.1: Summary of the Solution Mechanism

2.2 Surface Integral Equations

In this section, the basics of electromagnetic theory and the physical equivalence theorem are given as a fundamental theoretical background to obtain different types of surface integral equations (EFIE, MFIE, CFIE) for the solution of scattering problems from PEC bodies.

2.2.1 Basics of Electromagnetic Theory

The time domain, differential form of Maxwell's equations are given by

$$\nabla \times \boldsymbol{E}(\boldsymbol{r},t) \;=\; -\frac{\partial \boldsymbol{B}(\boldsymbol{r},t)}{\partial t} \tag{2.1}$$

$$\nabla \times \boldsymbol{H}(\boldsymbol{r},t) \;=\; \frac{\partial \boldsymbol{D}(\boldsymbol{r},t)}{\partial t} + \boldsymbol{J}(\boldsymbol{r},t) \tag{2.2}$$

$$\nabla \cdot \boldsymbol{D}(\boldsymbol{r},t) \;=\; \rho_e(\boldsymbol{r},t) \tag{2.3}$$

$$\nabla \cdot \boldsymbol{B}(\boldsymbol{r},t) \;=\; 0. \tag{2.4}$$

In (2.1)-(2.4), $\boldsymbol{E}(\boldsymbol{r},t)$ is the electric field intensity, $\boldsymbol{H}(\boldsymbol{r},t)$ is the magnetic field intensity, $\boldsymbol{D}(\boldsymbol{r},t)$ is the electric flux density, $\boldsymbol{B}(\boldsymbol{r},t)$ is the magnetic flux density, $\boldsymbol{J}(\boldsymbol{r},t)$ is the electric current density, $\rho_e(\boldsymbol{r},t)$ is the electric charge density and t represents the time dependence. In Cartesian coordinates the "Del" operator is expressed as

$$\nabla \;=\; \hat{x}\frac{\partial}{\partial x} + \hat{y}\frac{\partial}{\partial y} + \hat{z}\frac{\partial}{\partial z}, \tag{2.5}$$

and the position vector is given by

$$r \;=\; \hat{x}\boldsymbol{x} + \hat{y}\boldsymbol{y} + \hat{z}\boldsymbol{z}. \tag{2.6}$$

The electric current density is related to the electric charge density through the continuity equation as

$$\nabla \cdot \boldsymbol{J}(\boldsymbol{r},t) \;=\; \frac{-\partial \rho_e(\boldsymbol{r},t)}{\partial t}. \tag{2.7}$$

Flux densities and field intensities are related, in general, as

$$\boldsymbol{D}(\boldsymbol{r},t) \;=\; \bar{\boldsymbol{\epsilon}}(\boldsymbol{r})\cdot\boldsymbol{E}(\boldsymbol{r},t) \tag{2.8}$$

$$\boldsymbol{B}(\boldsymbol{r},t) \;=\; \bar{\boldsymbol{\mu}}(\boldsymbol{r})\cdot\boldsymbol{H}(\boldsymbol{r},t), \tag{2.9}$$

where $\bar{\boldsymbol{\epsilon}}(\boldsymbol{r})$ and $\bar{\boldsymbol{\mu}}(\boldsymbol{r})$ are permittivity and permeability values of media, respectively. In general, they are dyadic terms. Our focus in this study will be on simple media where the media are isotropic, linear, and homogeneous.

Therefore in a simple medium,

$$\bar{\boldsymbol{\epsilon}}(\boldsymbol{r}) \;=\; \epsilon\bar{\boldsymbol{I}} \qquad \bar{\boldsymbol{\mu}}(\boldsymbol{r}) \;=\; \mu\bar{\boldsymbol{I}},$$

where $\bar{\boldsymbol{I}}$ is the identity dyad given by

$$\bar{\boldsymbol{I}} \;=\; \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.10}$$

In frequency domain, using the $e^{j\omega t}$ time convention, with an angular frequency of $\omega = 2\pi f$, functions can be written in the following forms:

$$f(\boldsymbol{r},t) \;=\; Re\{f(\boldsymbol{r})e^{j\omega t}\} \tag{2.11}$$

$$\frac{\partial f(\boldsymbol{r},t)}{\partial t} \;=\; j\omega Re\{f(\boldsymbol{r})e^{j\omega t}\}. \tag{2.12}$$

So, using this convention, Maxwell's equations and continuity equation ((2.7)) can be written in the phasor domain as

$$\nabla\times\boldsymbol{E}(\boldsymbol{r}) \;=\; -j\omega\mu\boldsymbol{H}(\boldsymbol{r}) \tag{2.13}$$

$$\nabla\times\boldsymbol{H}(\boldsymbol{r}) \;=\; j\omega\epsilon\boldsymbol{E}(\boldsymbol{r}) + \boldsymbol{J}(\boldsymbol{r}) \tag{2.14}$$

$$\nabla\cdot\boldsymbol{D}(\boldsymbol{r}) \;=\; \frac{1}{\epsilon}\rho_e(\boldsymbol{r}) \tag{2.15}$$

$$\nabla\cdot\boldsymbol{B}(\boldsymbol{r}) \;=\; 0 \tag{2.16}$$

$$\nabla\cdot\boldsymbol{J}(\boldsymbol{r}) \;=\; -j\omega\rho_e(\boldsymbol{r}). \tag{2.17}$$

We can derive the well-known wave equations for field intensities by using Maxwell's equations in the absence of magnetic sources. By applying the following vector identity (curl identity) given by

$$\nabla \times \nabla \times \boldsymbol{f} \;=\; \nabla\nabla \cdot \boldsymbol{f} - \nabla^2 \boldsymbol{f} \tag{2.18}$$

to field intensities $\boldsymbol{E}(\boldsymbol{r})$ and $\boldsymbol{H}(\boldsymbol{r})$ in equations (2.13) and (2.14), respectively, we can obtain the wave equations as follows:

$$\nabla \times \nabla \times \boldsymbol{E}(\boldsymbol{r}) \;=\; -j\omega\mu \nabla \times \boldsymbol{H}(\boldsymbol{r}) \tag{2.19}$$

$$\nabla\nabla \cdot \boldsymbol{E}(\boldsymbol{r}) - \nabla^2 \boldsymbol{E}(\boldsymbol{r}) \;=\; -j\omega\mu(j\omega\epsilon \boldsymbol{E}(\boldsymbol{r}) + \boldsymbol{J}(\boldsymbol{r})) \tag{2.20}$$

$$\frac{1}{\epsilon}\nabla\rho_e(\boldsymbol{r}) - \nabla^2 \boldsymbol{E}(\boldsymbol{r}) \;=\; \omega^2\mu\epsilon \boldsymbol{E}(\boldsymbol{r}) - j\omega\mu \boldsymbol{J}(\boldsymbol{r}) \tag{2.21}$$

$$\nabla^2 \boldsymbol{E}(\boldsymbol{r}) + k^2 \boldsymbol{E}(\boldsymbol{r}) \;=\; j\omega\mu \boldsymbol{J}(\boldsymbol{r}) + \frac{1}{\epsilon}\nabla\rho_e(\boldsymbol{r}), \tag{2.22}$$

and

$$\nabla \times \nabla \times \boldsymbol{H}(\boldsymbol{r}) \;=\; j\omega\epsilon \nabla \times \boldsymbol{E}(\boldsymbol{r}) + \nabla \times \boldsymbol{J}(\boldsymbol{r}) \tag{2.23}$$

$$\nabla\nabla \cdot \boldsymbol{H}(\boldsymbol{r}) - \nabla^2 \boldsymbol{H}(\boldsymbol{r}) \;=\; j\omega\epsilon(-j\omega\mu \boldsymbol{H}(\boldsymbol{r})) + \boldsymbol{J}(\boldsymbol{r}) \tag{2.24}$$

$$-\nabla^2 \boldsymbol{H}(\boldsymbol{r}) \;=\; \omega^2\mu\epsilon \boldsymbol{H}(\boldsymbol{r}) + \nabla \times \boldsymbol{J}(\boldsymbol{r}) \tag{2.25}$$

$$\nabla^2 \boldsymbol{H}(\boldsymbol{r}) + k^2 \boldsymbol{H}(\boldsymbol{r}) \;=\; -\nabla \times \boldsymbol{J}(\boldsymbol{r}), \tag{2.26}$$

where $k = \omega\sqrt{\mu\epsilon} = 2\pi/\lambda$ is the wavenumber. In Cartesian coordinate system

$$\nabla^2 = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}. \tag{2.27}$$

Electric and magnetic field intensities can be written in terms of the vector and scalar potentials as follows [1]:

$$\boldsymbol{E}(\boldsymbol{r}) \;=\; -j\omega \boldsymbol{A}_m(\boldsymbol{r}) - \nabla\phi_e(\boldsymbol{r}) \tag{2.28}$$

$$\boldsymbol{H}(\boldsymbol{r}) \;=\; \frac{1}{\mu}\nabla \times \boldsymbol{A}_m(\boldsymbol{r}), \tag{2.29}$$

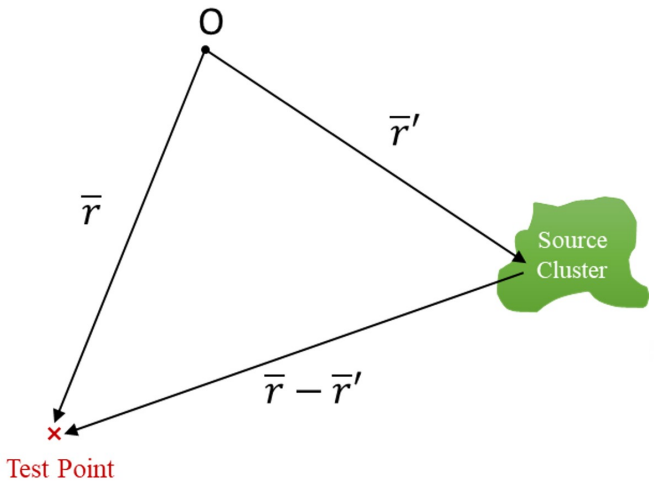


Figure 2.2: Test and Source Point Representation

where $\boldsymbol{A}_m$ and ϕ_e are the magnetic vector potential and the electric scalar potential, respectively. Then, using the Lorentz gauge given by

$$\nabla \cdot \boldsymbol{A}_m(\boldsymbol{r}) \;=\; -j\omega\epsilon\mu\phi_e(\boldsymbol{r}). \tag{2.30}$$

wave equations for potentials can obtained as

$$\nabla^2\phi_e(\boldsymbol{r}) + k^2\phi_e(\boldsymbol{r}) \;=\; -\frac{1}{\epsilon}\rho_e(\boldsymbol{r}) \tag{2.31}$$

$$\nabla^2\boldsymbol{A}_m(\boldsymbol{r}) + k^2\boldsymbol{A}_m(\boldsymbol{r}) \;=\; -\mu\boldsymbol{J}(\boldsymbol{r}). \tag{2.32}$$

The point source response of wave equations (i.e., the impulse response of an electromagnetic system in free-space) is the free-space Green's function, given by

$$g(\boldsymbol{r},\boldsymbol{r}') \;=\; \frac{e^{-jk|\boldsymbol{r}-\boldsymbol{r}'|}}{4\pi|\boldsymbol{r}-\boldsymbol{r}'|} \;=\; \frac{e^{-jkR}}{4\pi R} \qquad (R = |\boldsymbol{r}-\boldsymbol{r}'|) \tag{2.33}$$

where $\boldsymbol{r}$ is the test point and $\boldsymbol{r}'$ is the source point as depicted in Figure 2.2. Solution of (2.31) and (2.32) can be found by using the free-space Green's function (2.33). Expressions of $\boldsymbol{A}_m(\boldsymbol{r})$ and $\phi_e(\boldsymbol{r})$ for an arbitrary source distribution can be written as

$$\phi_e(\boldsymbol{r}) \;=\; \frac{1}{\epsilon}\int d\boldsymbol{r}' g(\boldsymbol{r},\boldsymbol{r}')\rho_e(\boldsymbol{r}') \tag{2.34}$$

$$\boldsymbol{A}_m(\boldsymbol{r}) \;=\; \mu\int d\boldsymbol{r}' g(\boldsymbol{r},\boldsymbol{r}')\boldsymbol{J}(\boldsymbol{r}'), \tag{2.35}$$

respectively. Substituting the potential expressions in (2.34) and (2.35) into (2.28) and (2.29), respectively, electric and magnetic field intensities can be written as

$$\boldsymbol{E}(\boldsymbol{r}) \;=\; -jk\eta \int d\boldsymbol{r}' \left(\boldsymbol{J}(\boldsymbol{r}') + \frac{1}{k^2} \nabla' \cdot \boldsymbol{J}(\boldsymbol{r}') \nabla \right) g(\boldsymbol{r}, \boldsymbol{r}') \tag{2.36}$$

$$\boldsymbol{H}(\boldsymbol{r}) \;=\; \int d\boldsymbol{r}' \nabla g(\boldsymbol{r}, \boldsymbol{r}') \times \boldsymbol{J}(\boldsymbol{r}'), \tag{2.37}$$

where $\eta = \sqrt{\dfrac{\mu_0}{\epsilon_0}}$ is the intrinsic impedance of free-space.

Electric and magnetic field intensities in (2.36) and (2.37) can be expressed in terms of integro-differential operators defined as

$$\boldsymbol{E}(\boldsymbol{r}) \;=\; \eta \boldsymbol{\mathcal{T}}\{\boldsymbol{J}\}(\boldsymbol{r}) \tag{2.38}$$

$$\boldsymbol{H}(\boldsymbol{r}) \;=\; \boldsymbol{\mathcal{K}}\{\boldsymbol{J}\}(\boldsymbol{r}), \tag{2.39}$$

where

$$\boldsymbol{\mathcal{T}}\{\boldsymbol{X}\}(\boldsymbol{r}) \;=\; -jk \int d\boldsymbol{r}' \left(\boldsymbol{X}(\boldsymbol{r}') + \frac{1}{k^2} \nabla' \cdot \boldsymbol{X}(\boldsymbol{r}') \nabla \right) g(\boldsymbol{r}, \boldsymbol{r}') \tag{2.40}$$

$$\boldsymbol{\mathcal{K}}\{\boldsymbol{X}\}(\boldsymbol{r}) \;=\; \int d\boldsymbol{r}' \boldsymbol{X}(\boldsymbol{r}') \times \nabla' g(\boldsymbol{r}, \boldsymbol{r}'). \tag{2.41}$$

2.2.2 Equivalence Principle

Electromagnetic scattering problems from homogeneous objects can be mathematically modeled by surface equivalence theorem (Huygen's principle) for the derivation of surface integral equations (SIEs) [1, 2]. According to the surface equivalence theorem, the actual sources inside a region of interest can be represented by equivalent sources on an imaginary boundary over the surface of the region. The proper current densities are defined on the imaginary boundary of the surface depicted as in Figure 2.3. Based on the uniqueness theorem, the equivalent current densities are calculated over the imaginary surface by the tangential components of the electric and magnetic field [1,6], while the boundary conditions are satisfied [41].

Furthermore, the physical equivalence theorem (which has a close relation with the surface equivalence theorem) can be employed to model the scattering problem as a radiation problem by defining the equivalent surface current densities (which are radiating to the region of interest) over the surface of the scatterer object. The equivalent current densities are calculated using the boundary conditions over the surface of the scatterer object.

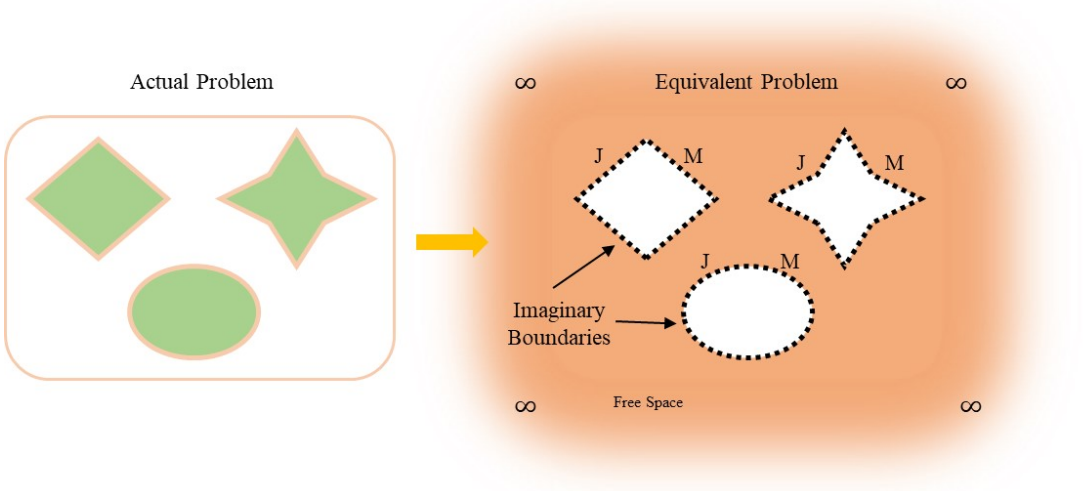


Figure 2.3: Definition of the actual problem and the equivalent problem

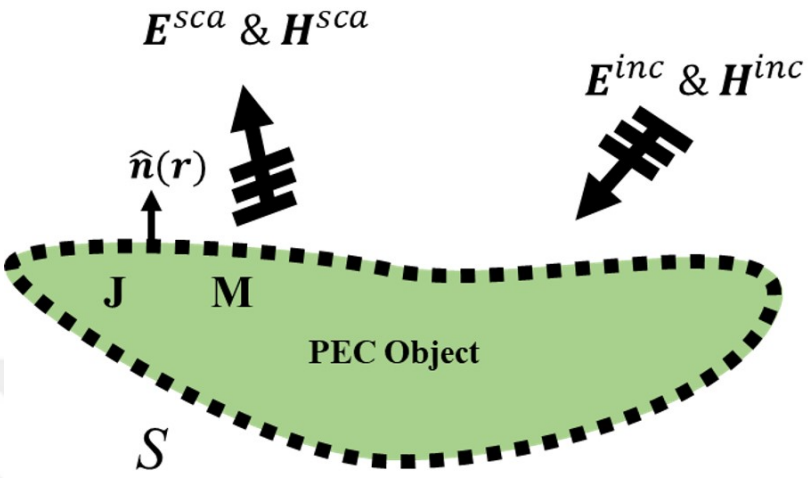


Figure 2.4: Surface boundary of a PEC object exposed to an electromagnetic radiation

Consider a PEC scatterer object bounded by a surface S depicted in Figure 2.4, where $\hat{\boldsymbol{n}}$ represents the unit outward surface normal vector; $\boldsymbol{J}$ and $\boldsymbol{M}$ are equivalent electric and magnetic current densities; $\boldsymbol{E}^{inc}$ and $\boldsymbol{E}^{sca}$ are incident and scattered electric fields, respectively, whereas $\boldsymbol{H}^{inc}$ and $\boldsymbol{H}^{sca}$ are incident and scattered magnetic fields, respectively. The equivalent surface currents can be related to the tangential components of the total fields at the surface boundary as,

$$\boldsymbol{J}(\boldsymbol{r}) \;=\; \hat{\boldsymbol{n}} \times \boldsymbol{H}(\boldsymbol{r}) \tag{2.42}$$

$$-\boldsymbol{M}(\boldsymbol{r}) \;=\; 0 \;=\; \hat{\boldsymbol{n}} \times \boldsymbol{E}(\boldsymbol{r}). \tag{2.43}$$

Expanding the total fields at the boundary as

$$\boldsymbol{H}(\boldsymbol{r}) \;=\; \boldsymbol{H}^{sca}(\boldsymbol{r}) + \boldsymbol{H}^{inc}(\boldsymbol{r})$$

$$\boldsymbol{E}(\boldsymbol{r}) \;=\; \boldsymbol{E}^{sca}(\boldsymbol{r}) + \boldsymbol{E}^{inc}(\boldsymbol{r}),$$

we obtain the tangential boundary equations of PEC materials as,

$$\boldsymbol{J}^{ind}(\boldsymbol{r}) - \hat{\boldsymbol{n}} \times \boldsymbol{H}^{sca}(\boldsymbol{r}) \;=\; \hat{\boldsymbol{n}} \times \boldsymbol{H}^{inc}(\boldsymbol{r}) \tag{2.44}$$

$$\hat{\boldsymbol{n}} \times \boldsymbol{E}^{sca}(\boldsymbol{r}) \;=\; -\hat{\boldsymbol{n}} \times \boldsymbol{E}^{inc}(\boldsymbol{r}), \tag{2.45}$$

where $\boldsymbol{J}^{ind}$ is the induced current on the surface of the material.

2.2.3 Formulation of SIEs

Electric field integral equation (EFIE) can be formed by enforcing the boundary condition for PEC materials in (2.45), and eliminating the $\boldsymbol{E}^{sca}$ dependence by writing it in terms of the induced current $\boldsymbol{J}^{ind}$ using the $\boldsymbol{\mathcal{T}}$ operator as in (2.36) resulting

$$\frac{-j}{k\eta}\left[\hat{\boldsymbol{t}}(\boldsymbol{r})\cdot\boldsymbol{E}^{inc}(\boldsymbol{r})\right] \;=\; \hat{\boldsymbol{t}}(\boldsymbol{r})\cdot\int_S d\boldsymbol{r}'\left(1+\frac{1}{k^2}\nabla'\cdot\boldsymbol{J}(\boldsymbol{r}')\nabla\right)g(\boldsymbol{r},\boldsymbol{r}'), \tag{2.46}$$

where $\hat{\boldsymbol{t}}(\boldsymbol{r})$ is the tangential surface normal. The same strategy is followed for the magnetic field expression. Magnetic field integral equation (MFIE) can be written by using the boundary condition (2.44) and separating the $\boldsymbol{\mathcal{K}}$ operator into its principal-value and limit parts [3] as

$$\hat{\boldsymbol{n}}(\boldsymbol{r})\times\boldsymbol{H}^{inc}(\boldsymbol{r}) \;=\; \frac{\Omega}{4\pi}\boldsymbol{J}(\boldsymbol{r})-\hat{\boldsymbol{n}}(\boldsymbol{r})\times\int_{S_{PV}} d\boldsymbol{r}'\boldsymbol{J}(\boldsymbol{r}')\times\nabla' g(\boldsymbol{r},\boldsymbol{r}'), \tag{2.47}$$

where, S_{PV} is the principal-value of the integral and Ω is the solid angle of the surface at the observation point.

To eliminate the internal resonance problem [2, 4], a linear combination of EFIE and MFIE is usually used, which is called as the combined field integral equation (CFIE) [4] given by

$$\text{CFIE} = \alpha\text{EFIE} + \frac{j}{k}(1-\alpha)\text{MFIE}, \tag{2.48}$$

where α can take values in the range of $0 \leq \alpha \leq 1$. However, the preferred choices are between $0.2 \leq \alpha \leq 0.5$. In order to balance MFIE term in CFIE, a factor of j/k is used [2, 3].

Matrix equations formed by the CFIE formulation commonly have better condition numbers [42] which leads to faster convergence in the iterative solutions.

2.3 Method of Moments

The unknown in the integral equation is the induced surface current density, which is a continuous function of space throughout the entire surface of the geometry. To solve it numerically, a common approach is to expand it in terms of known discrete functions, called basis/expansion functions, with unknown coefficients [2,6]. Then using the same number of testing functions, a linear system of equations can be obtained. In this way, the problem is converted into a matrix equation, which can be modeled and solved in the computer environment. This process can be formulated mathematically, known as the method of moments (MoM)as follows: Consider a general form of a linear problem as

$$\boldsymbol{\mathcal{L}}\{\boldsymbol{f}\}(\boldsymbol{r}) \;=\; \boldsymbol{g}(\boldsymbol{r}), \tag{2.49}$$

where $\boldsymbol{\mathcal{L}}$ is the linear operator applied on an unknown vector function $\boldsymbol{f}(\boldsymbol{r})$ to produce a known vector function $\boldsymbol{g}(\boldsymbol{r})$. $\boldsymbol{\mathcal{L}}$ is a linear combination of integro-differential operators $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{\mathcal{K}}$ whereas $\boldsymbol{f}(\boldsymbol{r})$ stands for the unknown current distribution ($\boldsymbol{J}(\boldsymbol{r})$) and $\boldsymbol{g}(\boldsymbol{r})$ is the excitation vector produced by the incident field. Unknown vector function $\boldsymbol{f}(\boldsymbol{r})$ can be expanded in a series of N weighted basis functions as

$$\boldsymbol{f}(\boldsymbol{r}) \;\approx\; \sum_{n=1}^{N} a[n]\boldsymbol{b_n}(\boldsymbol{r}), \tag{2.50}$$

where $\boldsymbol{b_n}$ $(n = 1, 2, ..., N)$ are the known basis vector functions and $a[n]$ is the unknown weighting coefficient of n^{th} basis function.
The inner product or moment between a basis function $\boldsymbol{f}_n(\boldsymbol{r}')$ and a testing/weighting function $\boldsymbol{t}_m(\boldsymbol{r})$ is expressed as [2]

$$\left\langle\, \boldsymbol{t}_m(\boldsymbol{r}), \boldsymbol{f}_n(\boldsymbol{r}') \,\right\rangle \;=\; \int \boldsymbol{t}_m(\boldsymbol{r}) \cdot \int \boldsymbol{f}_n(\boldsymbol{r}')d\boldsymbol{r}'d\boldsymbol{r}. \tag{2.51}$$

Inserting the expansion given in (2.50) into (2.49) and testing the resultant expression with a testing function $\boldsymbol{t_m}(\boldsymbol{r})$ $(m = 1, 2, ..., N)$ yields

$$\int d\boldsymbol{r}\boldsymbol{t_m}(\boldsymbol{r}) \cdot \sum_{n=1}^{N} a[n]\boldsymbol{\mathcal{L}}\{\boldsymbol{b_n}\}(\boldsymbol{r}) \;=\; \int d\boldsymbol{r}\boldsymbol{t_m}(\boldsymbol{r}) \cdot \boldsymbol{g}(\boldsymbol{r}). \tag{2.52}$$

Changing the order of summation and integration, a set of equations can be obtained as

$$\sum_{n=1}^{N} a[n] \int d\boldsymbol{r} \boldsymbol{t_m}(\boldsymbol{r}) \cdot \boldsymbol{\mathcal{L}}\{\boldsymbol{b_n}\}(\boldsymbol{r}) \;=\; \int d\boldsymbol{r} \boldsymbol{t_m}(\boldsymbol{r}) \cdot \boldsymbol{g}(\boldsymbol{r}), \tag{2.53}$$

which results in a $N \times N$ matrix equation

$$\sum_{n=1}^{N} a[n] Z[m,n] \;=\; w[m], \tag{2.54}$$

where

$$Z[m,n] \;=\; \int d\boldsymbol{r} \boldsymbol{t_m}(\boldsymbol{r}) \cdot \boldsymbol{\mathcal{L}}\{\boldsymbol{b_n}\}(\boldsymbol{r}) \tag{2.55}$$

$$w[m] \;=\; \int d\boldsymbol{r} \boldsymbol{t_m}(\boldsymbol{r}) \cdot \boldsymbol{g}(\boldsymbol{r}). \tag{2.56}$$

As a result, we obtain N linearly independent equations in (2.54) to calculate N unknown coefficients. In electromagnetic scattering problems, $\bar{\boldsymbol{Z}}[m,n]$ is called as the impedance matrix, and its elements are calculated via the discretization of surface formulations while the $\boldsymbol{w}[m]$ is called the excitation vector and is calculated by testing the incident fields on the surface.

2.3.1 RWG Basis Function

The unknown equivalent currents are usually expanded with Rao-Wilton-Glisson (RWG) basis functions [43]. Definition of an RWG function is given by [43]

$$\boldsymbol{b_n}(\boldsymbol{r})^{RWG} \;=\; \begin{cases} \dfrac{l_n}{2A_n^+}(\boldsymbol{r} - \boldsymbol{r}_n^+) = \dfrac{l_n}{2A_n^+}\boldsymbol{\rho_n^+}(\boldsymbol{r}), & \boldsymbol{r} \in S_n^+ \\ \dfrac{l_n}{2A_n^-}(\boldsymbol{r}_n^- - \boldsymbol{r}) = \dfrac{l_n}{2A_n^+}\boldsymbol{\rho_n^-}(\boldsymbol{r}), & \boldsymbol{r} \in S_n^- \\ 0, & \text{otherwise}, \end{cases} \tag{2.57}$$

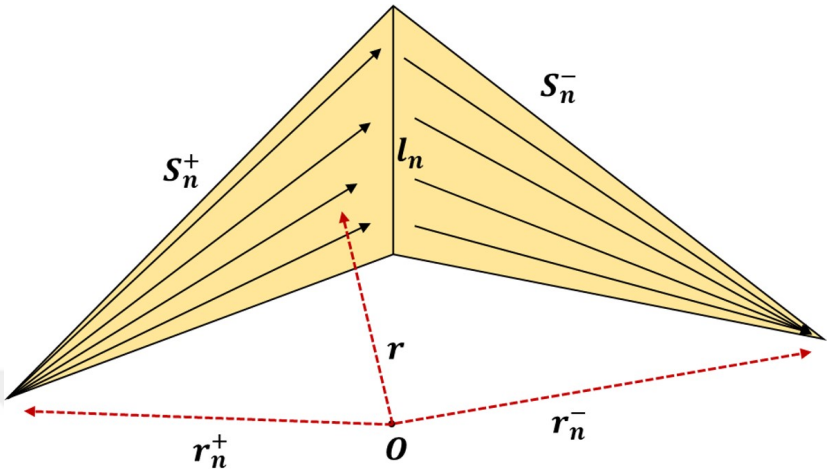


Figure 2.5: Definition of RWG Function

where A_n^+ and A_n^- are the areas of the triangles S_n^+ and S_n^-, respectively. Furthermore, l_n is the length of the common edge of the adjacent triangle couple, and $\boldsymbol{r}_n^+$ and $\boldsymbol{r}_n^-$ are the vertexes opposite to the common edge. One can see the illustration of an RWG function in Figure 2.5. Moreover, RWG function is divergence conforming and has a finite divergence everywhere, given by

$$\nabla \cdot \boldsymbol{b_n}(\boldsymbol{r})^{RWG} = \begin{cases} \dfrac{l_n}{A_n^+}, & \boldsymbol{r} \in S_n^+ \\ -\dfrac{l_n}{A_n^-}, & \boldsymbol{r} \in S_n^- \\ 0, & \text{otherwise.} \end{cases} \tag{2.58}$$

2.3.2 Galerkin Scheme and Matrix Elements of SIEs

In the context of this thesis, we apply a Galerkin method for the MoM solution of the surface integral equations, i.e., choosing the same set of basis and testing functions. We can expand the current density like in (2.50) as

$$\boldsymbol{J}(\boldsymbol{r'}) = \sum_{n=1}^{N} a[n]\boldsymbol{b_n}(\boldsymbol{r'}). \tag{2.59}$$

Inserting the expanded current density into EFIE and MFIE equations in (2.46) and (2.47), respectively, and then applying the testing procedure with the Galerkin scheme, impedance matrices can be obtained as

$$\begin{aligned}\bar{\boldsymbol{Z}}^{EFIE}[m,n] \;=\; & -j\omega\mu \int_{S_m} d\boldsymbol{r}\boldsymbol{t_m}(\boldsymbol{r}) \cdot \int_{S_n} d\boldsymbol{r}'\boldsymbol{b_n}(\boldsymbol{r}')g(\boldsymbol{r},\boldsymbol{r}'), \\ & + \frac{1}{j\omega\epsilon} \int_{S_m} d\boldsymbol{r}\nabla \cdot \boldsymbol{t_m}(\boldsymbol{r}) \cdot \int_{S_n} d\boldsymbol{r}' g(\boldsymbol{r},\boldsymbol{r}')\nabla' \cdot \boldsymbol{b_n}(\boldsymbol{r}')\end{aligned} \tag{2.60}$$

$$\begin{aligned}\bar{\boldsymbol{Z}}^{MFIE}[m,n] \;=\; & \frac{1}{2} \int_{S_m} d\boldsymbol{r}\boldsymbol{t_m}(\boldsymbol{r}) \cdot \boldsymbol{b_n}(\boldsymbol{r}) \\ & + \int_{S_m} d\boldsymbol{r}\boldsymbol{t_m}(\boldsymbol{r}) \cdot \hat{\boldsymbol{n}} \times \int_{PV,S_n} d\boldsymbol{r}'\boldsymbol{b_n}(\boldsymbol{r}') \times \nabla' g(\boldsymbol{r},\boldsymbol{r}'),\end{aligned} \tag{2.61}$$

and the right-hand-side excitation vectors are given by

$$\boldsymbol{w}^{EFIE}[m] \;=\; -\int_{S_m} d\boldsymbol{r}\boldsymbol{t_m}(\boldsymbol{r}) \cdot \boldsymbol{E}^{inc}(\boldsymbol{r}) \tag{2.62}$$

$$\boldsymbol{w}^{MFIE}[m] \;=\; -\int_{S_m} d\boldsymbol{r}\boldsymbol{t_m}(\boldsymbol{r}) \cdot \hat{\boldsymbol{n}} \times \boldsymbol{H}^{inc}(\boldsymbol{r}). \tag{2.63}$$

2.4 Numerical and Analytical Integration

It is important to choose effective numerical and analytical integration techniques compatible with the triangular domains to evaluate the integrals on the basis and test triangles to solve the integral equations accurately. In this thesis, the Gaussian quadrature technique is used for numerical integration; and required formulations in the literature are referenced for analytical integration.

2.4.1 Gaussian Quadrature Technique

An N point Gaussian quadrature rule for an integration of a function $f(\boldsymbol{r})$ over a triangle S with area A can be written as [2, 44]

$$\int\int_S d\boldsymbol{r} f(\boldsymbol{r}) \approx A \sum_{p=1}^{N} w_p f(\boldsymbol{r_p}), \tag{2.64}$$

where w_p is the integration weight. Locations of the integration points over the surface can be written as

$$\boldsymbol{r_p} = \alpha \boldsymbol{v_1} + \beta \boldsymbol{v_2} + \gamma \boldsymbol{v_3} \tag{2.65}$$

where $\boldsymbol{v_1}$, $\boldsymbol{v_2}$, $\boldsymbol{v_3}$ are the position vectors of vertexes of the triangle and α, β, γ are the simplex coordinates satisfying

$$\alpha + \beta + \gamma = 1. \tag{2.66}$$

As the number of integration points is increased, the accuracy of the integration usually increases. Number of integration points and the corresponding weights can be found in the literature [44].

2.4.2 Singularity Extraction of Integral Equations

When a test point and a basis point approach to each other, $R = |\boldsymbol{r} - \boldsymbol{r'}|$ goes to zero. Thus, the inner integrals of equations (2.60) and (2.61) diverge to infinity. Therefore, a proper singularity extraction technique must employed for EFIE (2.60) and MFIE (2.61).

For the EFIE part in (2.60), the following inner integrals can be divided as

$$\int_{S_n} d\boldsymbol{r'} \frac{e^{-jkR}}{R} = \int_{S_n} d\boldsymbol{r'} \frac{e^{-jkR} - 1}{R} + \int_{S_n} d\boldsymbol{r'} \frac{1}{R} \tag{2.67}$$

$$\int_{S_n} d\boldsymbol{r'} \boldsymbol{\rho_n}(\boldsymbol{r'}) \frac{e^{-jkR}}{R} = \int_{S_n} d\boldsymbol{r'} \boldsymbol{\rho_n}(\boldsymbol{r'}) \frac{e^{-jkR} - 1}{R} + \int_{S_n} d\boldsymbol{r'} \frac{\boldsymbol{\rho_n}(\boldsymbol{r'})}{R} \tag{2.68}$$

where

$$\lim_{R \to 0} \frac{e^{-jkR} - 1}{R} = -jk. \tag{2.69}$$

In (2.67) and (2.68), the first integrals on the right-hand-side of these equations can be evaluated numerically with the help of the limit case given by (2.69). On the other hand, the second integrals on the right-hand-side of these equations, an analytical calculation is needed [45, 46].

For the MFIE part, a similar procedure to that of EFIE can be applied. The inner integral in (2.61) can be written as

$$\begin{aligned} \int_{S_n} d\boldsymbol{r'} \left[(\boldsymbol{r} - \boldsymbol{r'}) \times \boldsymbol{\rho_n}(\boldsymbol{r'}) \right] \left[\frac{1 + jkR}{R^2} \right] g(\boldsymbol{r}, \boldsymbol{r'}) = \\ \boldsymbol{R}_n \times \int_{S_n} d\boldsymbol{r'} \boldsymbol{\rho_n}(\boldsymbol{r'}) \left[\frac{1 + jkR}{R^2} \right] g(\boldsymbol{r}, \boldsymbol{r'}), \end{aligned} \tag{2.70}$$

where

$$\boldsymbol{R}_n = \boldsymbol{r} - \boldsymbol{v}_n, \tag{2.71}$$

which yields

$$\boldsymbol{r} - \boldsymbol{r'} \;=\; \boldsymbol{R}_n - \boldsymbol{\rho_n}(\boldsymbol{r'}), \tag{2.72}$$

$$(\boldsymbol{r} - \boldsymbol{r'}) \times \boldsymbol{\rho_n}(\boldsymbol{r'}) \;=\; \boldsymbol{R}_n \times \boldsymbol{\rho_n}(\boldsymbol{r'}). \tag{2.73}$$

In (2.71), $\boldsymbol{v}_n$ is the opposite vertex of the RWG function edge. Now the integral on the right-hand-side of (2.70) can be expanded as

$$\begin{aligned} &\int_{S_n} d\boldsymbol{r'} \boldsymbol{\rho_n}(\boldsymbol{r'}) \left[\frac{1+jkR}{R^2}\right] g(\boldsymbol{r}, \boldsymbol{r'}) \;= \\ &\int_{S_n} d\boldsymbol{r'} \boldsymbol{\rho_n}(\boldsymbol{r'}) \frac{(1+jkR)e^{-jkR} - (1+\frac{1}{2}k^2R^2)}{R^3} + \boldsymbol{a}_n(\boldsymbol{r}) + \frac{k^2}{2}\boldsymbol{b}_n(\boldsymbol{r}), \end{aligned} \tag{2.74}$$

where

$$\boldsymbol{a}_n(\boldsymbol{r}) \;=\; \int_{S_n} d\boldsymbol{r'} \frac{\boldsymbol{\rho_n}(\boldsymbol{r'})}{R^3}, \tag{2.75}$$

$$\boldsymbol{b}_n(\boldsymbol{r}) \;=\; \int_{S_n} d\boldsymbol{r'} \frac{\boldsymbol{\rho_n}(\boldsymbol{r'})}{R}. \tag{2.76}$$

For the first term of (2.74), a similar limit case can be evaluated as in (2.69), given by

$$\lim_{R\to 0} \frac{(1+jkR)e^{-jkR} - (1+\frac{1}{2}k^2R^2)}{R^3} \;=\; \frac{-jk^3}{3}. \tag{2.77}$$

The remaining integrals $\boldsymbol{b}_n(\boldsymbol{r})$ and $\boldsymbol{a}_n(\boldsymbol{r})$ can be analytically calculated as explained in [45–47].

2.5 Multi-Level Fast Multiple Algorithm

The direct solution of the MoM matrix equation via matrix inversion has $O(N^2)$ memory and $O(N^3)$ time complexity (when solved directly). Using iterative solvers such as GMRES, time complexity can be reduced to $O(N^2)$ for each MVM, but it is still expensive when the number of unknowns, N, is large. Therefore, we accelerate this solution and also reduce the memory requirements using FMM and its multi-level version MLFMA. These accelerations start by separating the interactions between basis functions as near-zone and far-zone interactions. This separation can be expressed for the matrix equation as

$$Z^{Near} \cdot a + Z^{Far} \cdot a = w \tag{2.78}$$

The near-zone interactions in Z^{Near} are calculated directly and stored in the memory, whereas far-zone interactions are in Z^{Far}, and $Z^{Far} \cdot a$ is calculated on-the-fly without storing any values in the memory. Since FMM calculates the interactions in a group-by-group manner, the computational complexity reduces to $O(N^{3/2})$. This group-by-group manner interaction of FMM is illustrated in Figure 2.6.

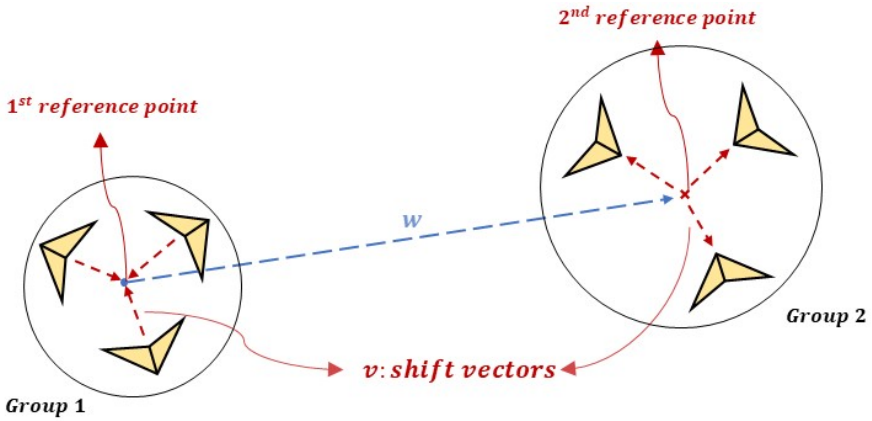


Figure 2.6: Group-by-group manner interactions for FMM

2.5.1 Addition Theorem for the Green's Function and Diagonalization

FMM and MLFMA are based on the Gegenbauer's addition theorem. The free-space Green's function can be rewritten as

$$g(\boldsymbol{r},\boldsymbol{r}') \;=\; \frac{e^{-jk|\boldsymbol{r}-\boldsymbol{r}'|}}{4\pi|\boldsymbol{r}-\boldsymbol{r}'|} \;=\; \frac{e^{-jk|\boldsymbol{w}+\boldsymbol{v}|}}{4\pi|\boldsymbol{w}+\boldsymbol{v}|}, \tag{2.79}$$

where $\boldsymbol{w}$ and $\boldsymbol{v}$ are the vectors that satisfy $|\boldsymbol{w}| = w > |\boldsymbol{v}| = v$. Then, the addition theorem can be used to express the Green's function as a sum of spherical waves given by

$$\frac{exp(-jk|\boldsymbol{w}+\boldsymbol{v}|)}{4\pi|\boldsymbol{w}+\boldsymbol{v}|} \;=\; \frac{-jk}{4\pi}\sum_{t=0}^{\infty}(-1)^t(2t+1)j_t(kv)h_t^{(1)}(kw)P_t(\hat{\boldsymbol{w}}\cdot\hat{\boldsymbol{v}}), \tag{2.80}$$

where j_t is the spherical Bessel function, $h_t^{(1)}$ is the spherical Hankel function of the first kind with order t, and P_t is the Legendre polynomial. Next, diagonalization process can be applied by writing the spherical waves in terms of plane-waves using the identity given by

$$j_t(kv)P_t(\hat{\boldsymbol{w}}\cdot\hat{\boldsymbol{v}}) \;=\; \frac{1}{4\pi(-j)^t}\int d^2\hat{\boldsymbol{k}}exp(-jk\hat{\boldsymbol{k}}\cdot\boldsymbol{v})P_t(\hat{\boldsymbol{k}}\cdot\hat{\boldsymbol{w}}), \tag{2.81}$$

where the integration is over the unit sphere, and $\hat{\boldsymbol{k}}$ is the radial unit vector. Now we can express the diagonalized form of the Green's function by inserting the identity given in (2.81) into (2.80) as

$$\frac{exp(-jk|\boldsymbol{w}+\boldsymbol{v}|)}{4\pi|\boldsymbol{w}+\boldsymbol{v}|} \;=\;$$

$$\frac{-jk}{4\pi}\sum_{t=0}^{\infty}(-1)^t(2t+1)h_t^{(1)}(kw)\frac{1}{4\pi(-j)^t}\int d^2\hat{\boldsymbol{k}}exp(-jk\hat{\boldsymbol{k}}\cdot\boldsymbol{v})P_t(\hat{\boldsymbol{k}}\cdot\hat{\boldsymbol{w}}) \tag{2.82}$$

$$\;=\; \frac{-jk}{(4\pi)^2}\int d^2\hat{\boldsymbol{k}}exp(-jk\hat{\boldsymbol{k}}\cdot\boldsymbol{v})\sum_{t=0}^{\infty}(-j)^t(2t+1)h_t^{(1)}(kw)P_t(\hat{\boldsymbol{k}}\cdot\hat{\boldsymbol{w}}). \tag{2.83}$$

In (2.83), we can define the shift and translation functions as

$$\beta(\hat{\boldsymbol{k}},\boldsymbol{v}) \;=\; exp(-jk\hat{\boldsymbol{k}}\cdot\boldsymbol{v}), \tag{2.84}$$

$$\alpha(\hat{\boldsymbol{k}},\boldsymbol{w}) \;=\; \sum_{t=0}^{\infty}(-j)^t(2t+1)h_t^{(1)}(kw)P_t(\hat{\boldsymbol{k}}\cdot\hat{\boldsymbol{w}}). \tag{2.85}$$

As a result, the diagonalized form of the Green's function can be read as

$$\frac{exp(-jk|\boldsymbol{w}+\boldsymbol{v}|)}{4\pi|\boldsymbol{w}+\boldsymbol{v}|} \;=\; \frac{-jk}{(4\pi)^2}\int d^2\hat{\boldsymbol{k}}\beta(\hat{\boldsymbol{k}},\boldsymbol{v})\alpha(\hat{\boldsymbol{k}},\boldsymbol{w}). \tag{2.86}$$

In numerical calculations, translation function is truncated at a finite value τ as [48]

$$\alpha_\tau(\hat{\boldsymbol{k}},\boldsymbol{w}) \;=\; \sum_{t=0}^{\tau}(-j)^t(2t+1)h_t^{(1)}(kw)P_t(\hat{\boldsymbol{k}}\cdot\hat{\boldsymbol{w}}), \tag{2.87}$$

where the excess bandwidth formula (EBF) given by

$$\tau \;\approx\; 1.73ka + 2.16(d_0)^{2/3}(ka)^{1/3} \tag{2.88}$$

is used for large boxes. In (2.88), a is the box size and d_0 is the number of accurate digits. Thus, the diagonalized form of the Green's function can be approximated as

$$\frac{exp(-jk|\boldsymbol{w}+\boldsymbol{v}|)}{4\pi|\boldsymbol{w}+\boldsymbol{v}|} \;\approx\; \frac{-jk}{(4\pi)^2}\int d^2\hat{\boldsymbol{k}}\beta(\hat{\boldsymbol{k}},\boldsymbol{v})\alpha_\tau(\hat{\boldsymbol{k}},\boldsymbol{w}). \tag{2.89}$$

Considering a scenario with multiple shifts as depicted in Figure 2.7, we can decompose the shift functions easily. Using the vector sum $\boldsymbol{r}-\boldsymbol{r'}=\boldsymbol{w}+\boldsymbol{v_1}+\boldsymbol{v_2}$, we can write the Green's function as

$$\frac{exp(-jk|\boldsymbol{w}+\boldsymbol{v_1}+\boldsymbol{v_2}|)}{4\pi|\boldsymbol{w}+\boldsymbol{v_1}+\boldsymbol{v_2}|} \;\approx\; \frac{-jk}{(4\pi)^2}\int d^2\hat{\boldsymbol{k}}\beta(\hat{\boldsymbol{k}},\boldsymbol{v_1}+\boldsymbol{v_2})\alpha_\tau(\hat{\boldsymbol{k}},\boldsymbol{w}) \tag{2.90}$$

$$= \frac{-jk}{(4\pi)^2}\int d^2\hat{\boldsymbol{k}}\beta(\hat{\boldsymbol{k}},\boldsymbol{v_1})\beta(\hat{\boldsymbol{k}},\boldsymbol{v_2})\alpha_\tau(\hat{\boldsymbol{k}},\boldsymbol{w}) \tag{2.91}$$

Furthermore, we can calculate the derivatives of the diagonalized Green's function as

$$\nabla g(\boldsymbol{r},\boldsymbol{r'}) \;\approx\; \frac{-jk}{(4\pi)^2}\int d^2\hat{\boldsymbol{k}}(-j\boldsymbol{k})\beta(\hat{\boldsymbol{k}},\boldsymbol{v})\alpha_\tau(\hat{\boldsymbol{k}},\boldsymbol{w}), \tag{2.92}$$

$$\nabla\nabla' g(\boldsymbol{r},\boldsymbol{r'}) \;\approx\; \frac{-jk}{(4\pi)^2}\int d^2\hat{\boldsymbol{k}}(\boldsymbol{k}\boldsymbol{k})\beta(\hat{\boldsymbol{k}},\boldsymbol{v})\alpha_\tau(\hat{\boldsymbol{k}},\boldsymbol{w}). \tag{2.93}$$

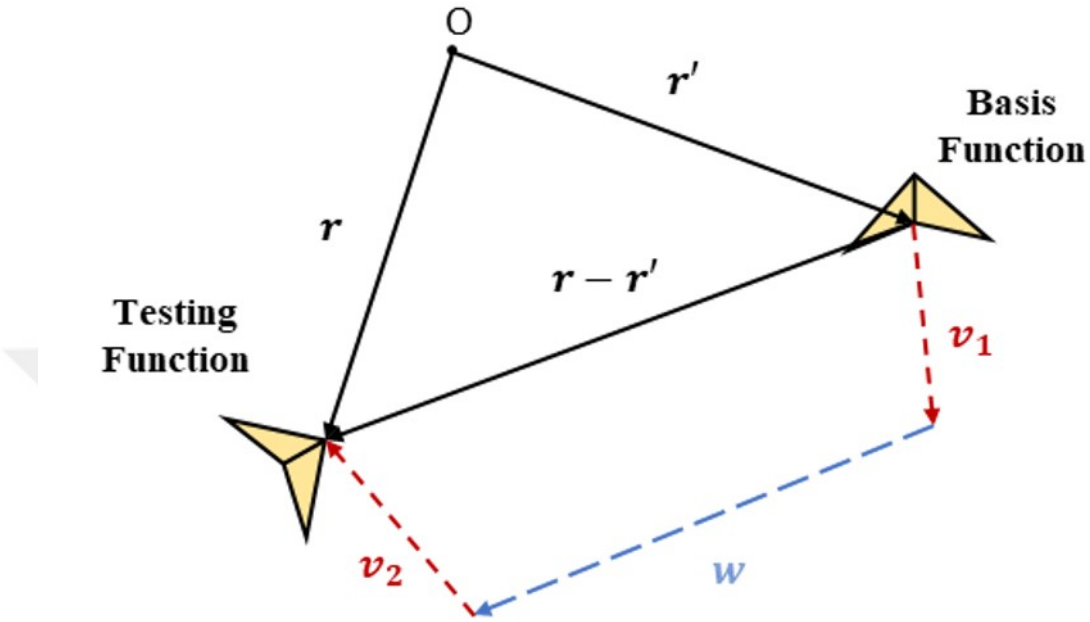


Figure 2.7: Interaction with Multiple Shift

In order to perform the integration of $\int d^2\hat{\boldsymbol{k}}$ over the unit sphere, a proper set of sampling points must be chosen. The following formulation is used as a rule of thumb to determine the number of sampling points in spherical coordinates:

$$S^{\theta} = \tau + 1 \tag{2.94}$$

$$S^{\phi} = 2(\tau + 1), \tag{2.95}$$

where $\tau \approx 1.73ka + 2.16(d_0)^{2/3}(ka)^{1/3}$. In this thesis, for the value of each sample, we use a regular sampling in the ϕ direction and a Gauss-Legendre distribution in the θ direction [17].

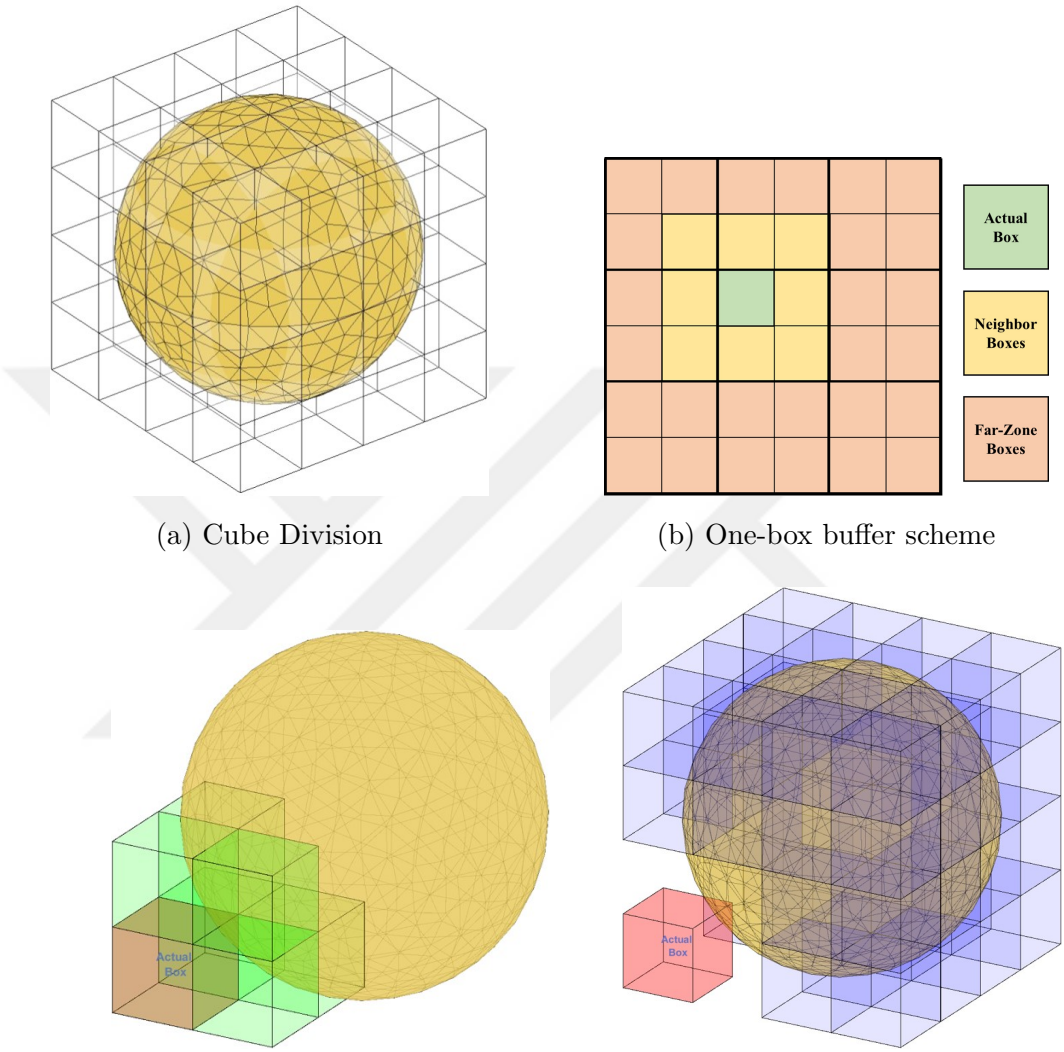
2.5.2 Clustering of FMM and One-Box Buffer Scheme

A cluster structure must be constructed for the FMM algorithm so that all interactions can be calculated in a group-by-group manner. For this purpose, a tree structure is formed, and the basis functions need to be distributed to groups in this scheme. The common approach is to put the discretized object in the smallest cubic box in which the object can be completely fit. Then, the cubic box is divided into sub-cubic boxes by halving the edges resulting in eight sub-cubes inside the largest cube. In this level, since all sub-cubes are neighbors of each other, the criterion of the Addition theorem (i.e., the translation vector must be larger than the sum of the shift vectors) cannot be satisfied. Thus, another division is needed, and this results in 64 sub-cubes inside the largest cube. Cube division of FMM cluster structure for a sphere object is shown in Figure 2.8(a). In this level, a one-box buffer scheme is used to determine the far-zone and near-zone interactions.

In the one-box buffer scheme, to determine the far-zone interactions of a box of interest (or group), the following two criteria must be satisfied.

1. Far-zone boxes are the children of the boxes that are neighbors of the parent box of the box of interest.
2. Far-zone boxes cannot be a neighbor box of the box of interest.

For near-zone interactions, near-zone boxes are the neighbor boxes of the box of interest. The one-box buffer scheme can be observed in Figure 2.8(b) for a two-dimensional case. A more detailed three-dimensional illustration of far-zone and near-zone boxes for a sphere object can be seen in Figure 2.8(c) and Figure 2.8(d).



(a) Cube Division

(b) One-box buffer scheme

(c) Near-zone boxes

(d) Far-zone boxes

Figure 2.8: FMM Clustering

MLFMA is the multi-level extension of FMM (which has been explained up to this point) [16, 49]. In MLFMA, computational complexity and memory complexity are further decreased to $O(NlogN)$ so that larger problems can be solved beyond the capacity of FMM. Multi-level group-by-group interactions can be seen in Figure 2.9. The algorithmic scheme of MLFMA is more complicated than FMM because of its multi-level scheme. There are three main stages in MLFMA, namely aggregation, translation, and disaggregation [3]. Since the box size is different for each level, the truncation numbers and the sample sizes are also different.

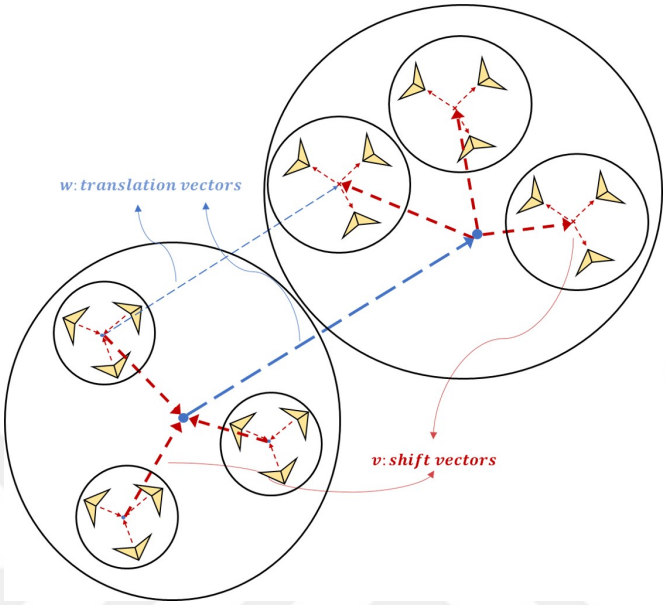
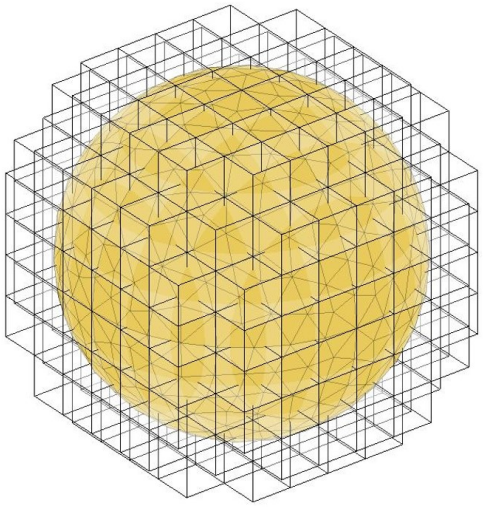


Figure 2.9: Multi-level group-by-group interaction

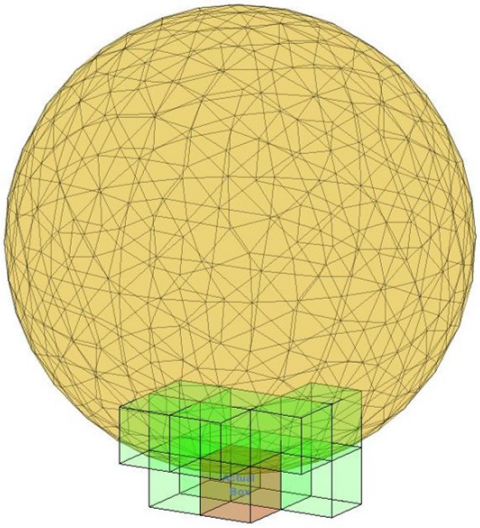
Thus to perform an accurate aggregation/disaggregation process between levels, an interpolation/anterpolation process is needed.

MLFMA tree structure is constructed in a similar way to that of FMM with the fact that the recursive division of cubes into sub-cubes (i.e., obtaining children boxes from a parent box) continues up to the desired level. After the division is completed, empty boxes are omitted, and only non-empty boxes are considered in the tree structure. This recursive division is illustrated in Figure 2.10(a) for a sphere geometry.

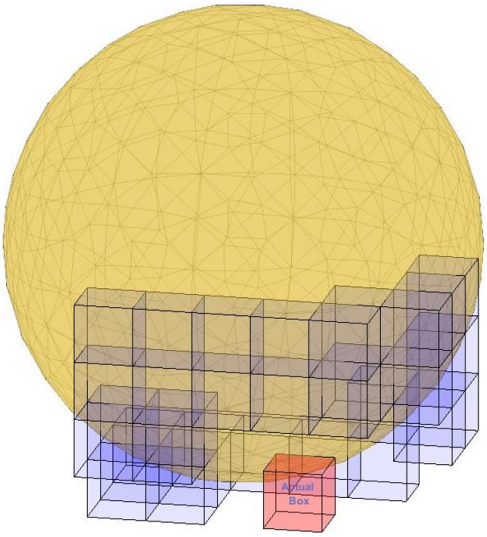
Once the tree structure is constructed, the stages of MLFMA can be employed depending on the child-parent relations, near-zone relations and far-zone relations. More specific, the parent-child relations (see Figure 2.11) are operated in the aggregation/disaggregation processes and the far-zone relations are operated in the translation process. The near-zone relations are operated not in MLFMA stages but in near-field calculations. A more detailed three-dimensional illustration of MLFMA near-zone and far-zone boxes for a sphere object can be seen in Figure 2.10(b) and Figure 2.10(c), respectively.



(a) Cube Division



(b) Near-zone boxes



(c) Far-zone boxes

Figure 2.10: MLFMA Clustering

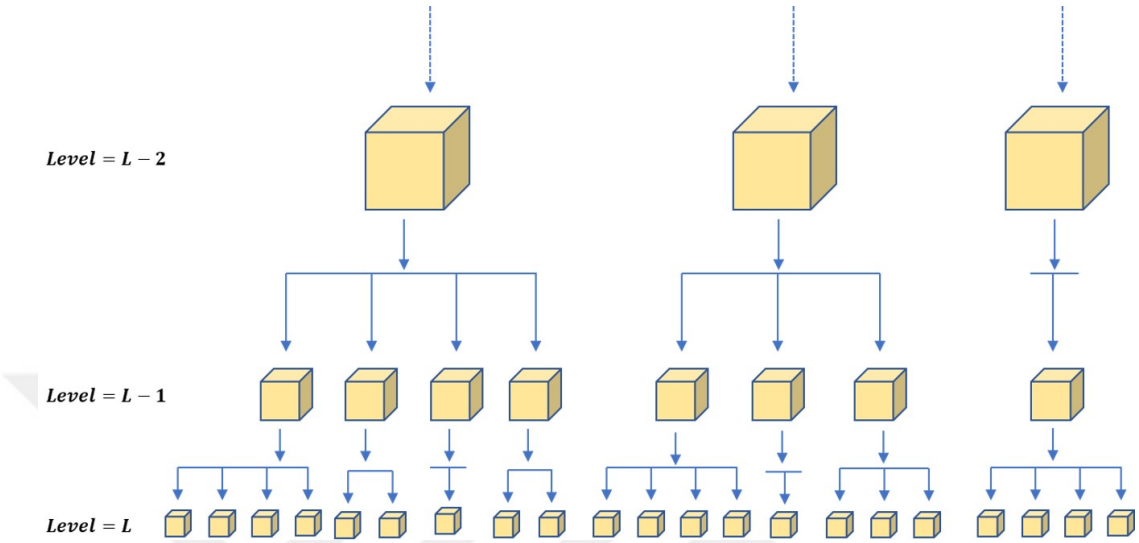


Figure 2.11: MLFMA child-parent relations of boxes

At the lowest level $\boldsymbol{L}$ of the tree structure, each non-empty box contains some discretized elements of the mesh model as depicted in Figure 2.12.

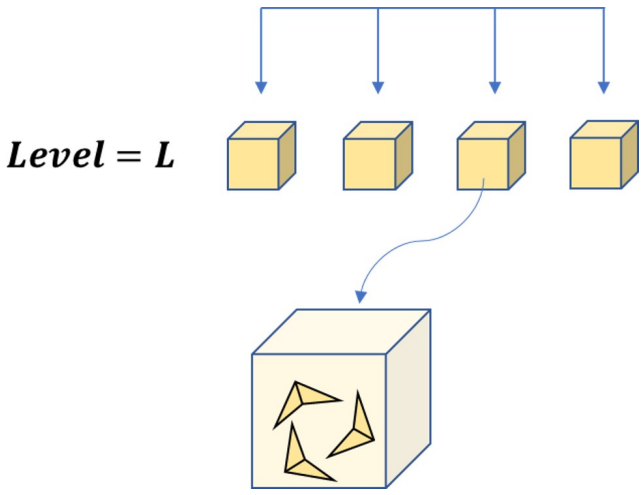


Figure 2.12: MLFMA child-parent relations of boxes

2.5.3 Stages of MLFMA

As mentioned before, group-by-group calculations for the far-zone interactions are performed in three stages: Aggregation, translation and disaggregation. These stages are briefly explained next.

2.5.3.1 Aggregation

In the aggregation process, radiated fields of boxes are calculated from the lowest level to the highest level. For this purpose, it is started from the lowest level boxes as seen in Figure 2.12. In the lowest level, all the radiation patterns of each basis function are multiplied with the corresponding element of the coefficient matrix $a[n]$ where the shifting function is directed to the center of the belonging box C. Aggregation at the lowest level can be written as

$$\bar{\boldsymbol{S}}_C(\hat{\boldsymbol{k}}, \boldsymbol{v_c}) \;=\; \sum_{\boldsymbol{b_n} \in C} a[n] \bar{\boldsymbol{S}}_n(\hat{\boldsymbol{k}}, \boldsymbol{v_c}), \tag{2.96}$$

where $\boldsymbol{v_c}$ is the vector from the center of the basis function (i.e., the center point of the common edge of the RWG function) to the center of the belonging box. $\bar{\boldsymbol{S}}_C$ is the radiated field of the belonging box C. For the higher levels, radiated fields at the center of the child box C' is multiplied by a diagonal shifting function, and thus aggregations from the center of the child box C' to the center of the parent box C can be written as

$$\bar{\boldsymbol{S}}_C(\hat{\boldsymbol{k}}, \boldsymbol{v_{cc'}}) \;=\; \sum_{C' \in C} \beta(\hat{\boldsymbol{k}}, \boldsymbol{v_{cc'}}) \bar{\boldsymbol{S}}_{C'}(\hat{\boldsymbol{k}}, \boldsymbol{v_{cc'}}), \tag{2.97}$$

where $\boldsymbol{v_{cc'}}$ is the vector from the center of the child box to the center of the parent box, β functions are the diagonal shift functions, and $\bar{\boldsymbol{S}}_{C'}$ is the radiated field of children boxes.

2.5.3.2 Translations

In the translation stage, all of the radiated fields are converted to receiving fields. Each box's radiated field is translated to all the boxes in its far-zone area at the

same level. This process is done for each level l of MLFMA. Translation of a radiated field is simply multiplying it with the translation operator α_τ for the level l as

$$\bar{\boldsymbol{G}}_{C'}(\hat{\boldsymbol{k}}, \boldsymbol{w_{c'c}}) \;=\; \sum_{C' \,\in\, F\{C\}} \alpha_\tau^L(\hat{\boldsymbol{k}}, \boldsymbol{w_{c'c}}) \bar{\boldsymbol{S}}_C, \tag{2.98}$$

where $F\{C\}$ represents all the far-zone boxes of C. $\bar{\boldsymbol{G}}_{C'}$ is the translated pattern (or receiving pattern) of box C', $\boldsymbol{w_{c'c}}$ is the vector from the center of box C to center of the box C' and α_τ is the diagonal translation function.

2.5.3.3 Disaggregation

In the disaggregation stage, receiving fields are calculated from the highest level to the lowest level. While doing this, a disaggregated field from the parent box to the child box is added to the current receiving field of the child box. This process is continued until we arrive to the lowest level. Disaggregation of sub-levels can be written as

$$\bar{\boldsymbol{G}}^{+}_{C'}(\hat{\boldsymbol{k}}, \boldsymbol{v_{c'c}}) \;=\; \bar{\boldsymbol{G}}_{C'} + \beta(\hat{\boldsymbol{k}}, \boldsymbol{v_{c'c}}) \bar{\boldsymbol{G}}_C \tag{2.99}$$

where $\boldsymbol{v_{c'c}}$ is the vector from the center of parent box C to the center of child box C' and $\bar{\boldsymbol{G}}_{C'}$ is the receiving field of the child box before the disaggregation and $\bar{\boldsymbol{G}}^{+}_{C'}$ is the receiving field of the child box after the disaggregation. Finally, for the lowest level, receiving fields of boxes $\bar{\boldsymbol{G}}^{+}_{C'}$ are distributed to the testing functions $\boldsymbol{t_m}$ by taking the dot product of the corresponding receiving patterns and then the spherical integration is done. Resulting number is the m^{th} element of the matrix-vector-multiplication as

$$\boldsymbol{V}^{Far}[m] \;=\; \sum_{n=1}^{N} Z^{Far}[m,n] a[n] \;=\; \left(\frac{-jk}{4\pi}\right)^2 \int d^2\hat{\boldsymbol{k}} \bar{\boldsymbol{F}}_m(\hat{\boldsymbol{k}}, \boldsymbol{v_{c'}}) \cdot \bar{\boldsymbol{G}}^{+}_{C'}, \tag{2.100}$$

where $\boldsymbol{v_{c'}}$ is the vector from the center of the box C' to the center of the testing function $\boldsymbol{t_m}$.

Chapter 3

MLFMA Implementation on GPU Systems

In this chapter, firstly, a background information about the GPU hardware infrastructure is given. Then, MLFMA is investigated to determine which parts of the MLFMA can be executed on GPU. Next, bottlenecks and possible solutions for GPU implementation are discussed and different implementation approaches are proposed for possible solutions. Finally, preliminary results for different implementation approaches are given.

3.1 Graphics Processing Units

In today's technology, GPUs are widely used in many research areas such as image processing, bio-informatics, numerical applications, mathematical modeling, medical imaging, virtual reality, and so forth [21, 23, 38, 50]. The performance needs of engineering applications generally have high-performance requirements, and these requirements cannot be satisfied by a single-core or even multi-core CPU systems most of the time. GPU becomes a good alternative to CPU for the solution of dense computational problems with its outperforming

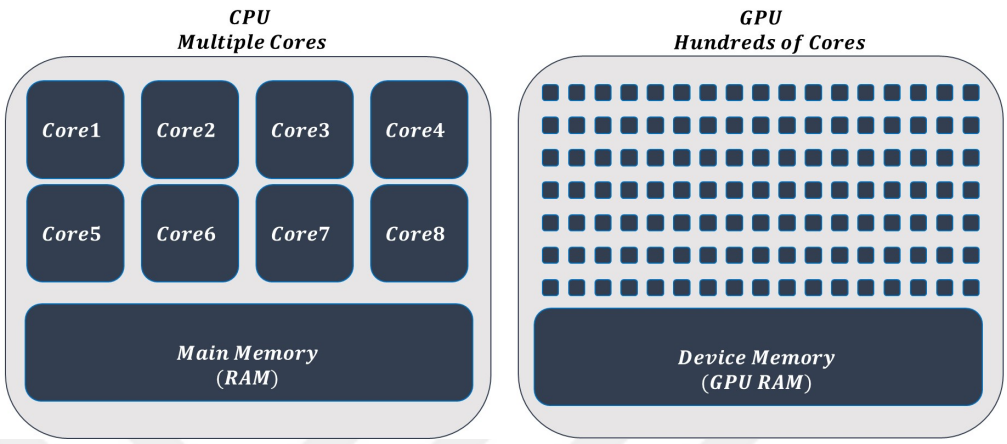


Figure 3.1: CPU and GPU cores

memory bandwidth, and floating-point performance [23, 51, 52]. General-purpose computation on GPU (GPGPU) has been getting easier over the last years in various fields by the development of new programming systems compatible with the GPU coding [20, 22, 53].

Graphics processing units (GPUs) are Single-Instruction-Multiple-Data (SIMD) processors [20, 23, 51, 52]. They can do the same operation on multiple data in parallel by its multi-core structure. There are thousands of small, efficient cores in the GPU, whereas the CPU has only a few cores as shown in Figure 3.1. GPU cores can process the same operation on massive data simultaneously, giving high computational throughput for parallel computing. The important point is that different cores cannot do different operations in parallel. Thus, algorithms compatible with the SIMD structure must be chosen for an efficient GPU implementation. Otherwise, GPU implementation may result in performance degradation [29, 38].

A GPU consists of threads, blocks, and grids. Thread is the smallest processing unit of the GPU, and different threads can calculate the same operation in parallel on a different part of a data set. Blocks are made up of a group of threads, and grids are made up of a group of blocks [39, 54]. An overview of GPU structure is in Figure 3.2.

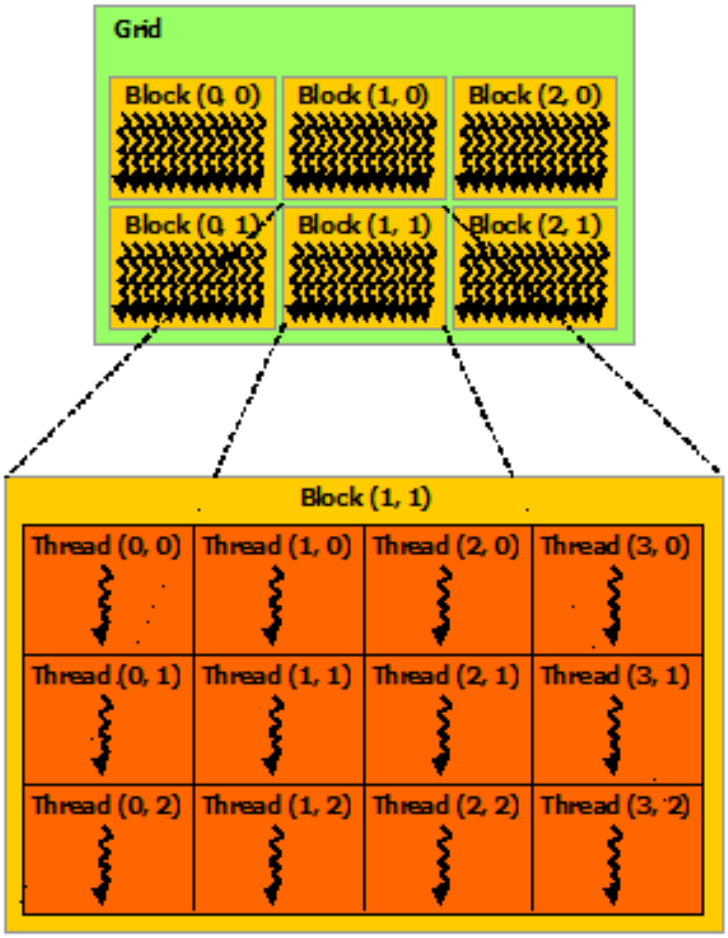


Figure 3.2: GPU architecture overview [55]

There are different types of memories in the GPU structure. These are register memory, local memory, shared memory, global (or device) memory, constant memory, and texture memory [55]. Each memory type has different functionality during the GPU process, but within the scope of this thesis, the use of different memories is not investigated or implemented. The device memory can be read from all threads, and also each thread can write to this memory. Device memory is the biggest memory type and determines the capacity of the problem to be calculated. Thus, we only consider the device memory capacity of the GPU cards in this work.

Data transmissions between host (CPU) and device (GPU) are done through the PCI-E (Peripheral Component Interconnect-Express) bus [38, 52]. Depending on

the bandwidth of the memory bus, data transmission speeds can change both in the upstream (CPU to GPU) and downstream (GPU to CPU) directions. In the literature, pinned-memory or global memory techniques are used to enhance the data transmission speeds concerning the memory need of the desired algorithm.

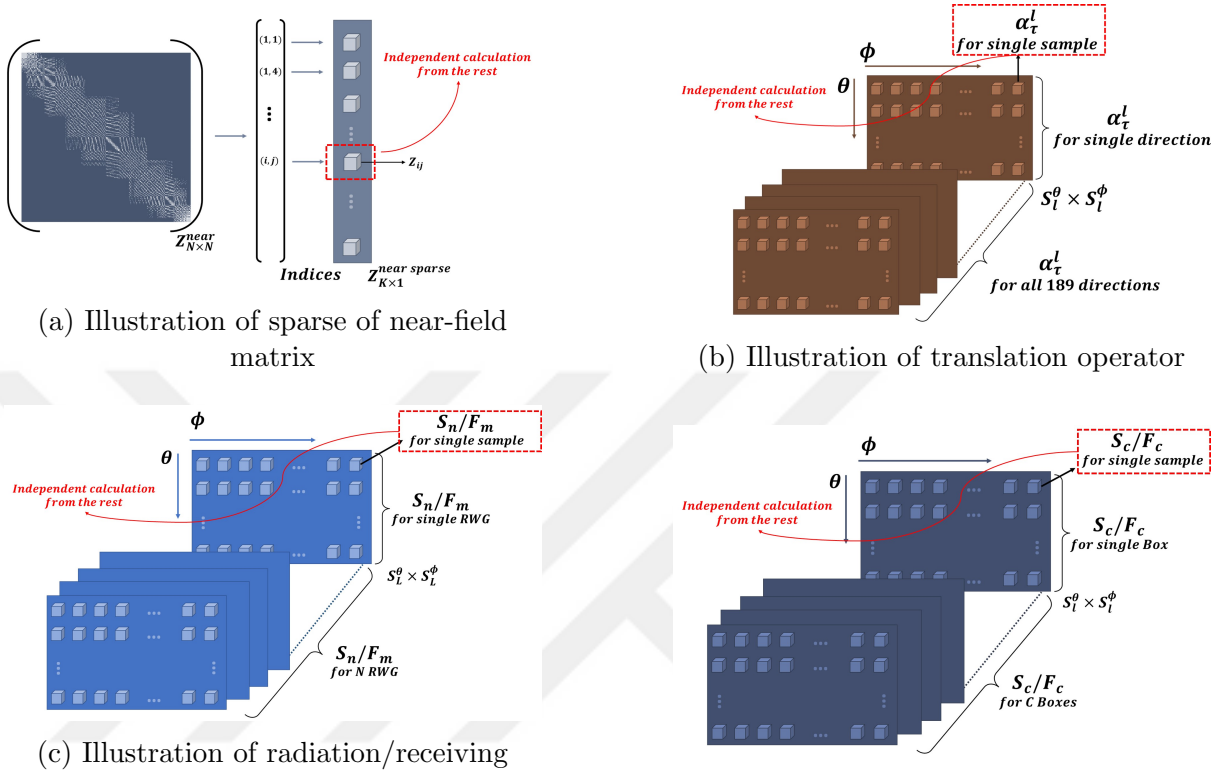
Floating-point precision is another important issue in GPU implementation. Double-precision arithmetic gives accurate results, but its computational cost is also high. Single-precision arithmetic is faster than double-precision but has lower accuracy. According to the numbers given by manufacturers [56] processing power capacities of GPUs can vary up to 30 times between single and double precision arithmetic. Therefore, we also implemented some single-precision GPU implementations to see this performance gap, however, our results were not accurate (because of the lower-precision) and also we were not able to reach the claimed efficiencies in the literature [56, 57].

3.2 Implementation of MLFMA on GPU

The hardware background of GPU structure (such as SIMD) is explained in the previous section. In this section, each step of MLFMA is analyzed to check their compatibility for SIMD structure in order to decide whether they can be implemented on GPU or not. However, SIMD compatibility is not the only but a must criteria for an efficient GPU implementation. Thus, in the following sections, other requirements of GPU implementation are further investigated for MLFMA as well.

The scheme of MLFMA is commonly considered in two main parts: setup and solution. In the setup part, there are two main operations, namely the calculation of $\bar{\boldsymbol{Z}}^{Near}$ matrix and the calculation of translation operators α_τ. In the solution part, $\bar{\boldsymbol{Z}}^{Far} \cdot \boldsymbol{a}$ is calculated in the stages of aggregation, translation and disaggregation.

Throughout this section, we assume an L level MLFMA problem consists of N



(a) Illustration of sparse of near-field matrix

(b) Illustration of translation operator

(c) Illustration of radiation/receiving patterns

(d) Illustration of higher level fields

Figure 3.3: MLFMA data structures of different variables

unknowns for a sample calculation. Only the θ and ϕ components exist for the radiation/receiving patterns/fields (depicted in Figure 3.3(c) and Figure 3.3(d), respectively) in the spherical coordinate system depending on the formulations [3]. Sample drawings of the data structures of radiated/receiving patterns/fields, translation operator, and near-field matrix are shown in Figure 3.3. We can also observe the independent unit of calculations for each data structure from Figure 3.3.

In the setup part, $\bar{\boldsymbol{Z}}^{Near}$ matrix is directly calculated by the standard MoM procedure and stored in the main memory. $\bar{\boldsymbol{Z}}^{Near}$ matrix is a sparse matrix and can be modeled as a column vector with sparse matrix indices as shown in Figure 3.3(a). The size of the near-field matrix (denoted as K) can change depending on the total level L of MLFMA. Each of the K elements can be calculated independently from the rest [3, 4]. Thus, assembling the near-field matrix is perfectly compatible with SIMD structure of GPU.

Another calculation of setup part is the translation operator α_τ for some level l. Since the maximum number of possible far-field interactions for an arbitrary box is 189 in MLFMA clustering structure, α_τ has a size of $S_l^\theta \times S_l^\phi \times 189$ at level l as depicted in Figure 3.3(b). S_l^θ and S_l^ϕ are the numbers of θ and ϕ samples, respectively, at level l. Each sample of α_τ is independently calculated, and hence α_τ is also compatible with SIMD.

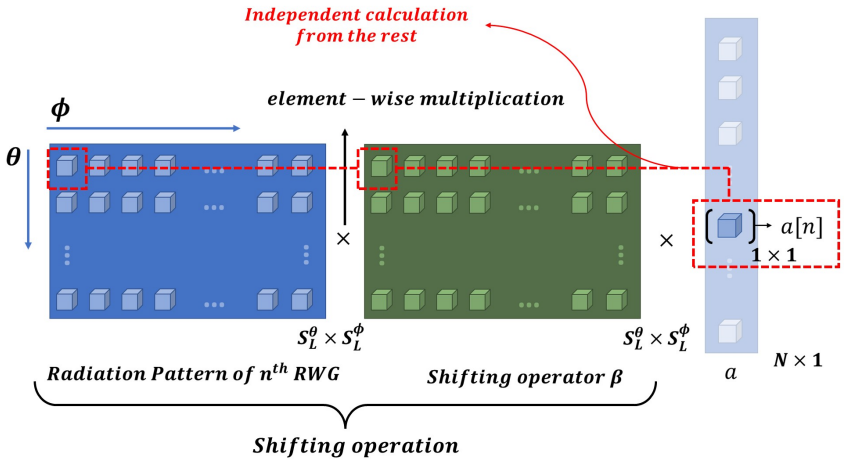
In the solution part, stages of MLFMA are employed, and operational structure of each stage are examined separately. In the lowest-level aggregation stage, the radiation pattern of each RWG function $\bar{\boldsymbol{S}}_n$ is calculated at each sampling point for θ and ϕ components separately. In total, entire radiation pattern for a single component is an $S_L^\theta \times S_L^\phi \times N$ array as shown in Figure 3.3(c), where N is the total number of RWG functions. Then, radiation pattern of each RWG function are multiplied with a shifting function and with the corresponding element of a vector, respectively. Both the assembling of radiation pattern and multiplication operations (shown in Figure 3.4(a)) are the independent unit of works, and hence can be implemented on SIMD structure.

In the higher-level aggregations, radiated fields of each box are interpolated and shifted, respectively. At some sub-level l with total non-empty box of C, entire radiated field for a single component is an $S_l^\theta \times S_l^\phi \times C$ array depicted in Figure 3.3(d). Since interpolation process is a matrix-vector-product as shown in Figure 3.4(b), both the interpolation and the shifting operations can be implemented on SIMD structure.

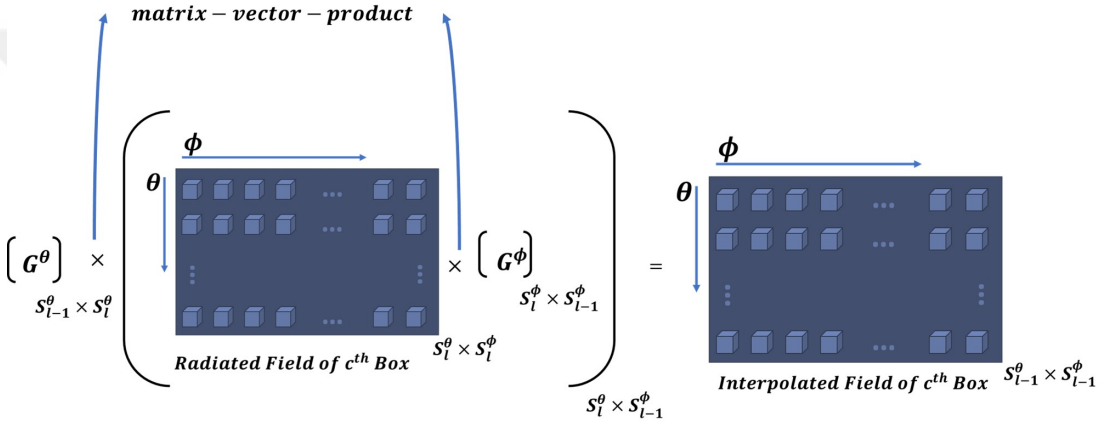
In the translation stage, at some level l with total non-empty box of C, there are $S_l^\theta \times S_l^\phi \times C$ radiated fields. These radiated fields are translated into receiving fields towards the far-field boxes by multiplying all samples with the corresponding translator operator. So if we assume that there are L times far-field interactions in total at that level l, then there are totally $S_l^\theta \times S_l^\phi \times L$ multiplication operations for one component of the fields, which can be done in parallel on the GPU as shown in Figure 3.4(c).

In the higher-level disaggregations, receiving fields of each box are shifted and anterpolated, respectively. At some sub-level l with total non-empty box of C, entire receiving field is of $S_l^\theta \times S_l^\phi \times C$ array. Anterpolation is a similar matrix-vector-product to that of interpolation. Hence, both anterpolation and shifting multiplication operations are compatible with SIMD similar to the higher-level aggregations.

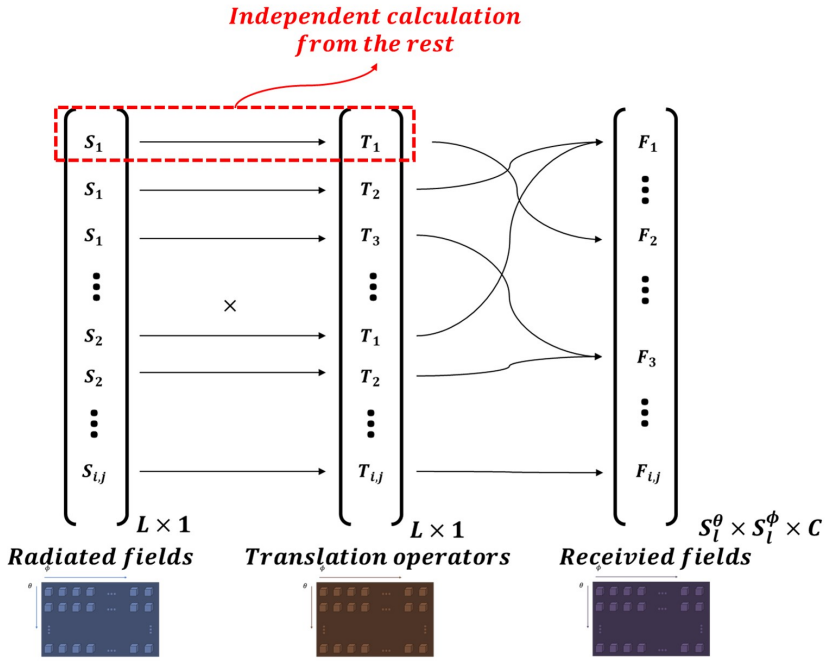
In the lowest level disaggregation, receiving pattern of each RWG function $\bar{\boldsymbol{F}}_m$ is calculated at each sampling point. In total, entire receiving pattern is an $S_L^\theta \times S_L^\phi \times N$ array as shown in Figure 3.3(c). Then, the dot product of each receiving pattern with receiving fields of including box is taken. Since dot product is an SIMD operation, both the dot product and receiving pattern assembling can be implemented on SIMD structure as well.



(a) Illustration of lowest level aggregation



(b) Illustration of interpolation process at sub-level aggregation stage



(c) Illustration of translation process

Figure 3.4: Illustration of MLFMA stages

3.3 Bottlenecks of GPU and Possible Remedies for the Solution of Large-Scale Problems

In this section, the bottlenecks of GPU implementation is explained in detailed and then possible remedies and some implementation approaches are offered to eliminate the bottlenecks of GPU while concerning the overall efficiency by considering different implementation approaches.

3.3.1 Bottlenecks of GPU

The GPU implementation have some restrictions when solving real-life large-scale EM problems. Firstly, the data transmission has a cost and is also limited to the bandwidth of the communication bus [38, 52, 58]. Data transmission cost needs to be considered both in the upstream and downstream directions. The data package (to be processed) needs to be sent to the GPU, processed, and returned back to the CPU. All the transmission cost on the way is a critical issue in the implementation.

The second issue is the low memory capacity of GPU cards. Global memory capacities of GPUs are relatively lower than the main memory (RAM) of the computer, and this capacity problem restricts the maximum data size that can be processed in the GPU at a time [53]. In our concern, this memory limit bounds the electrical size of the problem that can be solved in a single-GPU card [29,30].

Some other critics are the memory initialization, and device initialization costs that are independent of the size of the processed data or the memory bandwidths [29, 38]. These initialization costs occur whenever a GPU kernel is called.

Finally, since there are many types of memories in the GPU, internal delays between different types of memories can also occur during the implementations [53]. These bottlenecks are considered, and possible solutions are offered next.

3.3.2 Possible Remedies and Implementation Approaches

Data transmission delays are inevitable costs, and there is no way to eliminate them in a GPU implementation. To decrease the impact of transmission delays on the overall efficiency, the time spent on computation in the GPU must dominate the time spent transferring the data from and to the CPU. In other words, the operational weight of the computational process done in the GPU must be intensive [29, 57], so that the time saved in the computation is much higher than the time lost in the transmission. The time efficiency of a GPU implementation must always be checked for low-operational-weight functions. Otherwise, performance degradation can happen in the overall execution [38]. In Figure 3.5, the general scheme of GPU implementation is showed. The desired timing analysis of a GPU implementation can be written as

$$t_{total}^{GPU} = t_{process}^{GPU} + t_{delay}^{GPU} < t_{process}^{CPU} \tag{3.1}$$

$$t_{process}^{GPU} + t_{delay}^{GPU} < t_{process}^{CPU} \tag{3.2}$$

$$t_{delay}^{GPU} < (t_{process}^{CPU} - t_{process}^{GPU}) = \text{time saved by GPU}, \tag{3.3}$$

whereas the undesired timing condition is given by

$$t_{delay}^{GPU} \geq (t_{process}^{CPU} - t_{process}^{GPU}) = \text{time saved by GPU}, \tag{3.4}$$

where

$$t_{delay}^{GPU} = t_{delay}^{GPU,initialization} + t_{delay}^{GPU,internal\ memory\ com} + t_{delay}^{GPU,data\ transmission}. \tag{3.5}$$

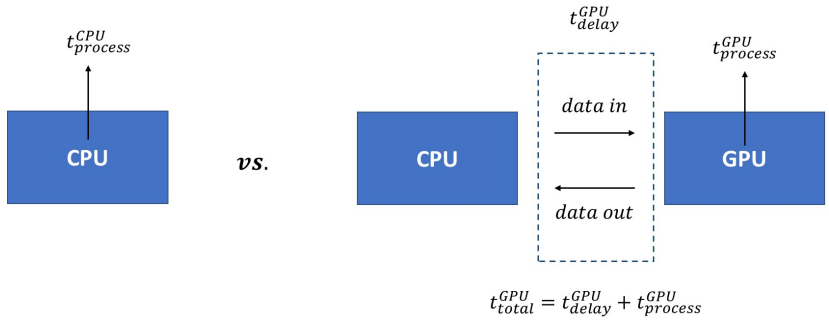


Figure 3.5: General scheme of GPU implementation

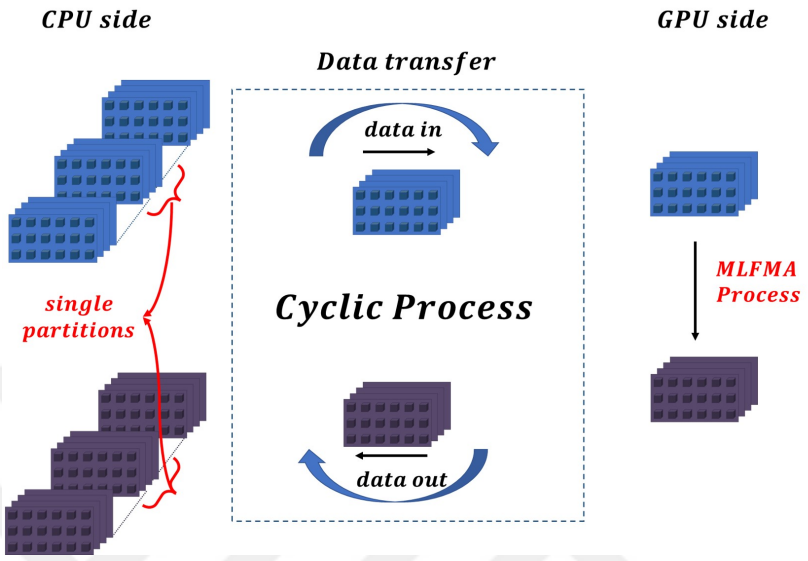
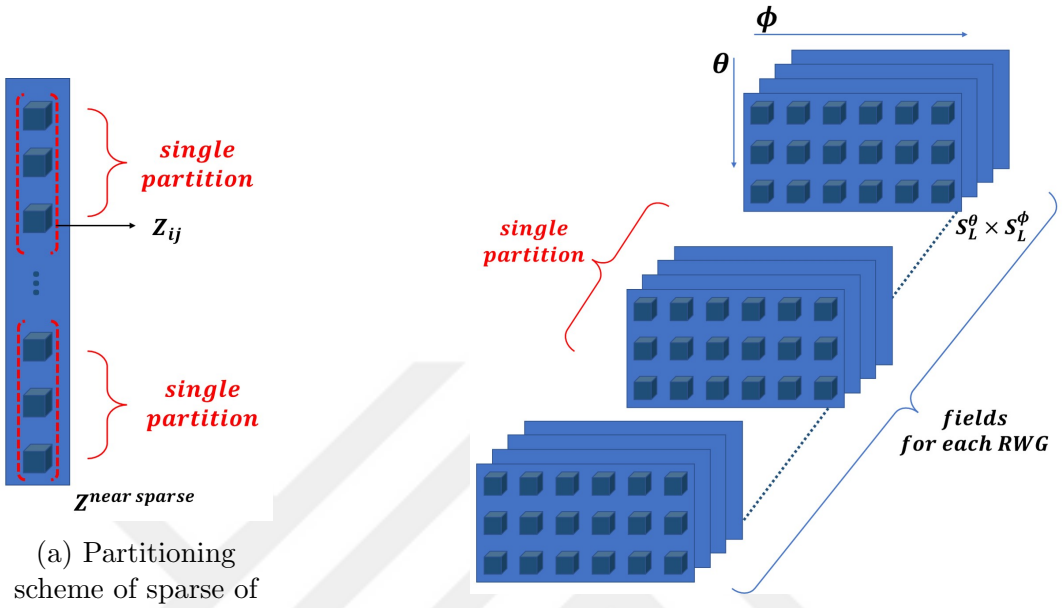


Figure 3.6: Cyclic process of the proposed GPU implementation

Low memory capacity is another bottleneck and a strict restriction of GPU implementation for large-scale EM problems because the size of data in MLFMA exceeds the memory capacity of the GPU at some point. Therefore, we propose a data partitioning scheme to solve this problem. In this scheme, the entire big data is not calculated at once on the GPU, but instead, little parts of it are calculated in order. The desired output data is split into small partitions that fit into GPU memory, and those partitions are calculated one by one in a cyclic process. For this purpose, input data is also split into small partitions. At each cycle, the corresponding partition of input data is sent to GPU, and the corresponding partition of the output data is calculated in the GPU and then sent back to CPU. This cyclic process (see Figure 3.6) continues until the entire data is calculated. Calculation of each element of MLFMA data is an independent unit of work as it is shown in Figure 3.3 and Figure 3.4. Hence the calculation of each partition is also independent of each other.

A similar approach of matrix partitioning is applied for a MoM application in the literature by splitting the dense MoM matrix into little segments and solving each segment separately in a single-GPU card [37]. In addition, an analogy can be established between the out-of-core method, which is used for reducing RAM usage by using hard-drive disks. In the out-of-core technique, big size of data is



(a) Partitioning scheme of sparse of near-field matrix

(b) Partitioning scheme of fields

Figure 3.7: Partitioning scheme of different data structures of MLFMA

stored in hard-drive sources and is fetched whenever it is needed [59, 60]. Figure 3.7 shows how different data types are split into partitions in our proposed scheme.

There can be different approaches when determining the size of each partition. The size of the partition is inversely proportional to the number of total cycles. As the partition size increases, the total number of cycles decreases and vice versa. So, larger partition size results in fewer cycles and hence fewer initialization delays occur at each data transmission event. However, the PCI-E bus also has a capacity, and thus large partition sizes can also overtake the PCI-E bus limit and/or internal memory limits, resulting in additional transfer delays. These delay considerations may be a crucial issue, especially when the operational weight of the problem is small for some cases. We examine each approach separately.

The first approach for determining the partition size is to maximize the partition size, which minimizes the number of total cycles and device initialization effects. For the first approach, the following procedure can be followed:

1. Available memory of the GPU must be determined. Before fetching the

available memory, the GPU device must always be reset to release the non-use memory.

2. Required memory for the specific mathematical process (such as near-field matrix calculation, radiation pattern calculation, etc.) must be calculated. Required memory is the sum of the size of input parameters to be transferred, intermediate parameters to be calculated on the fly, and the resulting output data.

3. Total number of cycles can be calculated as,

$$\text{Total cycles} = \text{ceil}(\text{Required Memory}/\text{Available Memory}).$$

In the above, the ceil function is used to round the fractional number to up because the number of total cycles must be an integer value while the memory requirement is safely protected. In addition, since GPU cards are not dedicated just for our implementations and their memory is also used by the operating systems; a further safety condition (such as 10% or 30%) can be set to prevent any out-of-memory conditions during the process.

$$\text{Total cycles} = \text{ceil}(1.3*(\text{Required Memory}/\text{Available Memory})).$$

After doing many implementation trials with GPU to test the proposed scheme, we experienced that 30% safety condition is better regarding the effects of other applications which can request GPU RAM.

4. Size of each partition (or number of elements calculated in a single cycle or the step size) can be determined as,

$$\text{step size} = \text{floor}(\text{Total number of elements}/\text{Total cycles}).$$

The floor function is used to round the fractional number down if a remainder occurs. If a remainder value occurs, it can be calculated at the last cycle in the implementation.

5. Remainder can be detected as follows

Remainder = remainder(Total number of elements/Total cycles).

The second approach can be increasing the number of total cycles and thus decreasing the size of each partition to check whether an optimized package size exists or not. A balance point between the PCI-E bus limit and internal memory limit may be detected. So, this approach needs to be tested for the same mathematical algorithm with different values of package sizes for a particular hardware system. For the second approach, there is no specific formulation. Optimum package size for a particular hardware system can be determined by a benchmark tool that repeats the same mathematical calculation for different package sizes and saves the timings. The resulting timing graph can give an idea of whether an optimum package size or an optimum region for the package size exists or not. The preliminary results for testing these approaches are given in the following chapter.

Chapter 4

Numerical Results & Discussion

In this chapter, numerical results of this study are presented for various GPU implementations of MLFMA. Before giving the results of GPU implementations, some validation results are given to show the accuracy of our methods. After that, MLFMA GPU implementation results are presented in two parts: first, the preliminary results for testing different implementation approaches of GPU implementation, and second the actual GPU implementations of MLFMA on various geometries such as different sizes of spheres and some real-life geometries (missile, aircraft).

All GPU implementations are compared with single-thread CPU implementations, which is considered as a reference to assess the efficiency and to calculate the speed-up values.

For all methods (MoM and/or MLFMA), solutions of matrix equations are obtained by using the iterative solver GMRES [40] with the tolerance value (relative residual error) of 10^{-3}. Inner and outer iteration parameters are set to 1000 and 1, respectively. No preconditioning techniques are used. Double-precision arithmetic is used.

Throughout this chapter, for the scattered field comparisons with Mie solutions,

percentage norm errors are calculated to show the accuracy of the solutions. Percentage norm errors are calculated as

$$\%\ Norm\ Error = \frac{\sqrt{\sum_{\theta} \left[\boldsymbol{E}_{MIE}(\theta) - \boldsymbol{E}_{scat}(\theta) \right]^2}}{\sqrt{\sum_{\theta} \left[\boldsymbol{E}_{MIE}(\theta) \right]^2}} \times 100 \tag{4.1}$$

In all implementations, the geometries are discretized with triangular patches compatible with RWG functions with mesh size of approximately $\lambda/10$ as a rule of thumb [3].

4.1 Validation Results

In this section, validation results of MLFMA are presented. First, the scattered electric fields from two different PEC spheres are compared for MoM, MLFMA, and Mie series solution (the analytic solution). Secondly, the scattered electric fields from two different PEC cubes are compared for MoM and MLFMA.

In this validation section, all geometries are illuminated with a linearly polarized plane-wave. Plane-wave is unit magnitude, x-polarized ([1,0,0]) and propagating along $+z$-direction ([0,0,1]) as shown in Figure 4.1 and Figure 4.3. The resulting scattered fields are calculated along the $\phi = 0^{\circ}$ cut (xz-plane) where θ values vary from 0 to 360 degrees with 1 degree spacing as shown in Figure 4.1 and Figure 4.3 (see the circular red lines). For both methods, CFIE formulation is used with $\alpha = 0.5$. For MoM interactions and MLFMA near-field interactions, 3-point testing and 4-point basis functions are used for the integrations over the triangles. For MLFMA far-interactions, 3-point testing and 3-point basis functions are used for the integrations over the triangles.

4.1.1 Sphere Analyses

In this part, two different PEC sphere geometries are used with diameters of 1λ ($0.3m$) and 4λ ($0.3m$) operating at 1 GHz and 4 GHz frequencies, respectively.

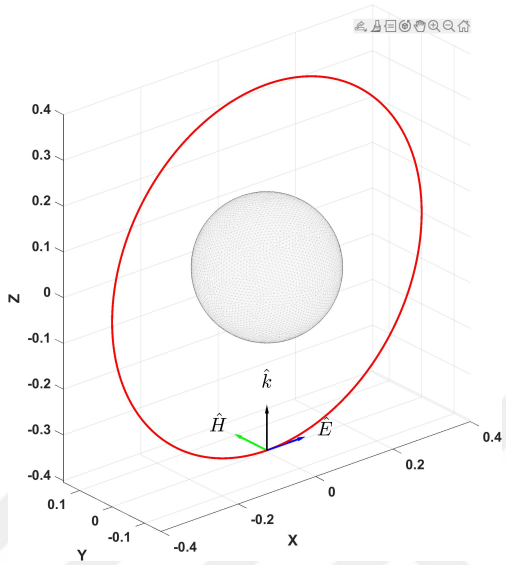
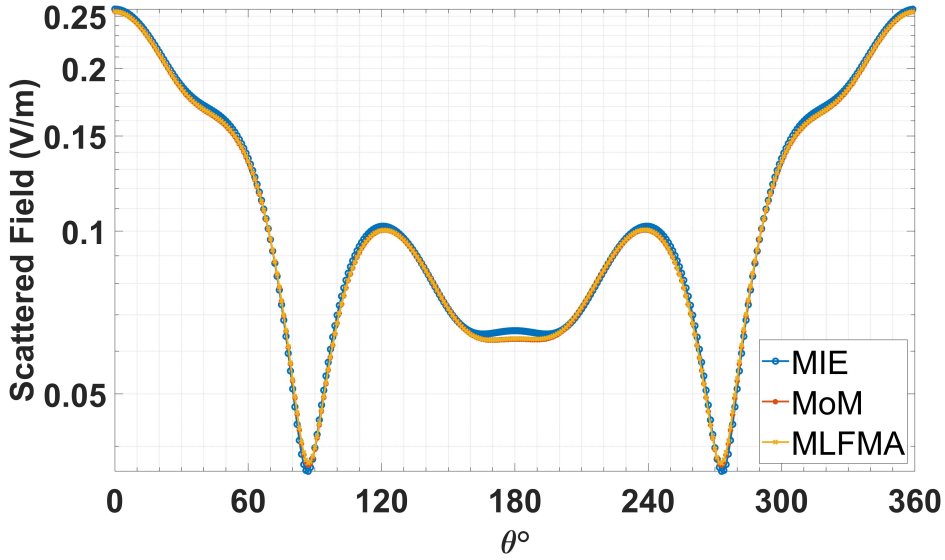


Figure 4.1: Illumination of a PEC sphere geometry by an x-polarized ([1 0 0]), +z-propagated ([0 0 1]) plane wave.

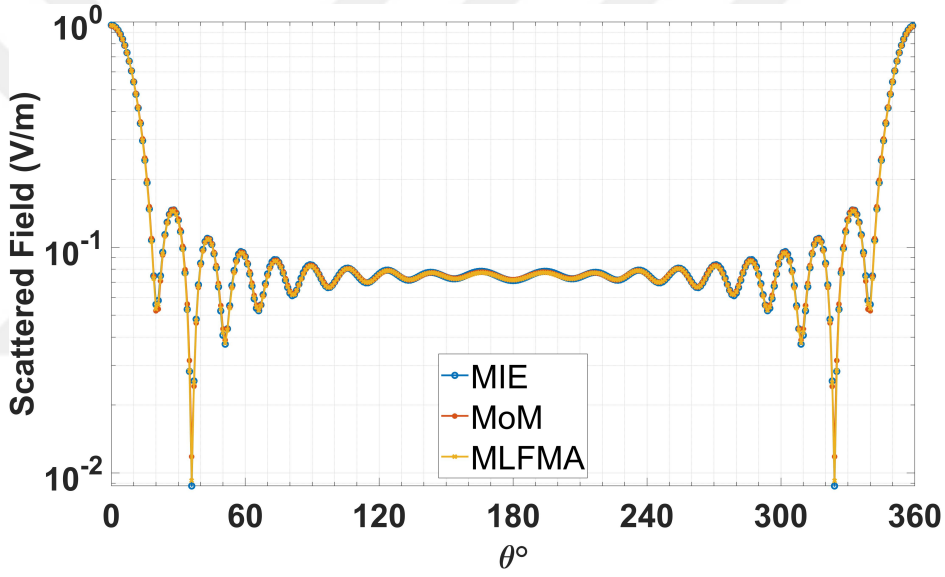
The 1λ sphere geometry has 854 triangular patches and 1281 RWG functions, whereas the 4λ sphere geometry has 13040 triangular patches and 19560 RWG functions.

For the 1 GHz analysis, the resulting scattered fields can be observed from Figure 4.2(a) and the induced current on the geometry is in Figure 4.2(c). The norm errors with respect to the Mie solution at 1 GHz are 1.5314% and 1.4707% for MoM and MLFMA, respectively. The error decreases as we increase the number of unknowns (discretization error).

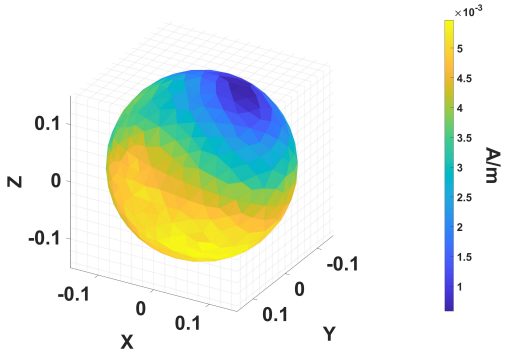
For the 4 GHz analysis, the resulting scattered fields can be observed from Figure 4.2(b) and the induced current on the geometry is in Figure 4.2(d). The norm errors with respect to the Mie solution at 4 GHz are 0.8956% and 0.3796% for MoM and MLFMA, respectively. Numerical results (and the norm errors) show that we have a good agreement with the Mie series solution, which is analytical and is used as a reference.



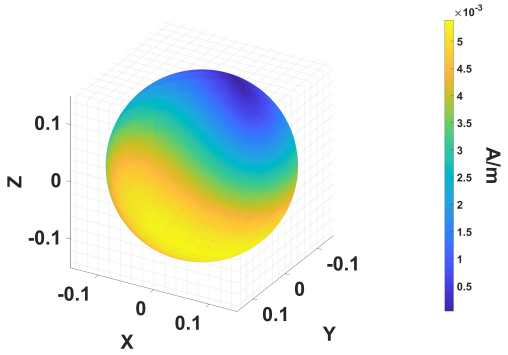
(a) Scattered E-fields from 1λ sphere geometry.



(b) Scattered E-fields from 4λ sphere geometry.



(c) Induced currents on 1λ sphere.



(d) Induced currents on 4λ sphere.

Figure 4.2: The comparison of scattered fields (MoM and MLFMA) with the Mie series solution ((a),(b)) and induced currents ((c),(d)) for different electrical sizes of spheres used for validation analyses.

4.1.2 Cube Analyses

In this part, two different PEC cube geometries are used with edge lengths of 1λ $(0.3m)$ and 4λ $(0.3m)$ operating at 1 GHz and 4 GHz frequencies, respectively. The 1λ cube geometry has 1470 triangular patches and 2205 RWG functions, whereas the 4λ cube geometry has 22244 triangular patches and 33366 RWG functions.

For the 1 GHz analysis, the resulting scattered fields can be observed from Figure 4.4(a), and the induced current on the geometry is in Figure 4.4(c). The norm error between MoM and MLFMA solutions at 1 GHz is 0.5759%. Note that MoM solution is used as a reference solution.

For the 4 GHz analysis, the resulting scattered fields can be observed from Figure 4.4(b), and the induced current on the geometry is in Figure 4.4(d). The norm error between MoM and MLFMA solutions at 4 GHz is 0.9729%. As seen from the results (scattered field and induced current) very good agreement between MoM and MLFMA results is obtained.

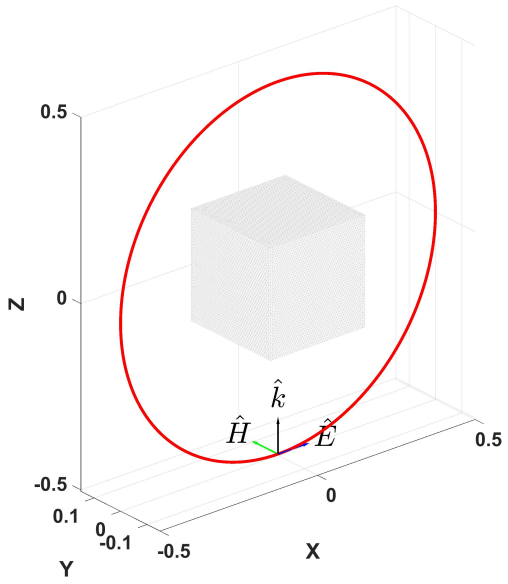
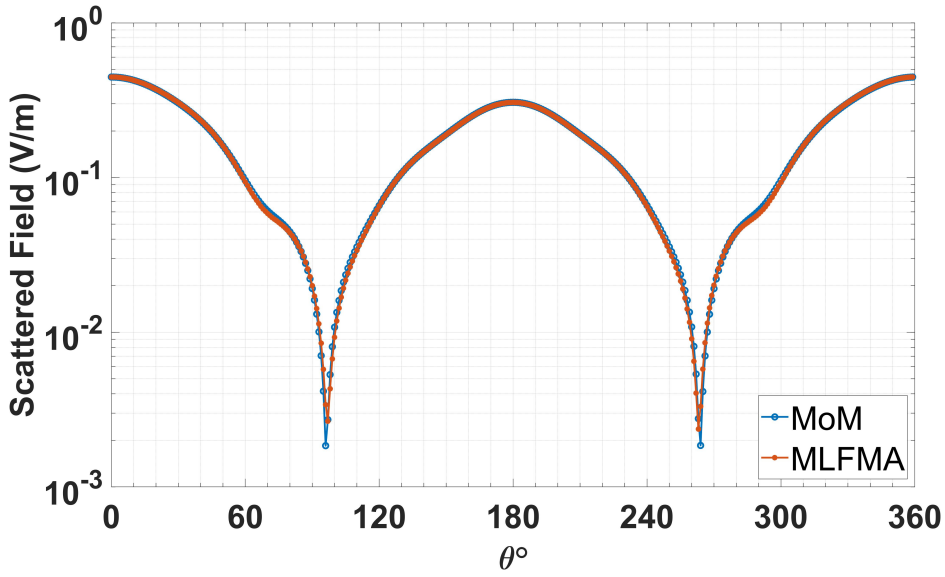
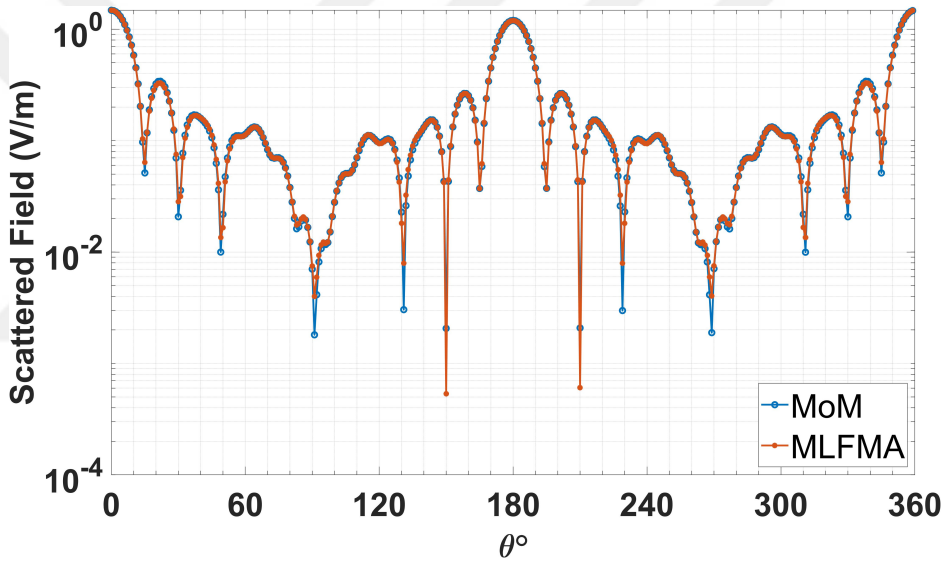


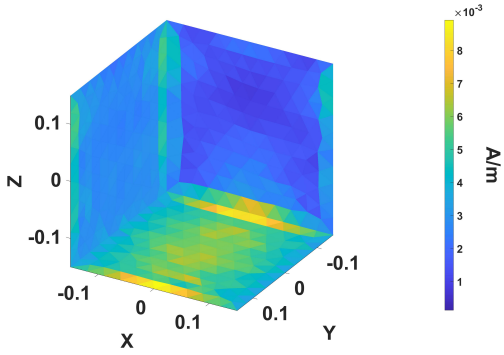
Figure 4.3: Illumination of a PEC cube geometry by an x-polarized ([1 0 0]), +z-propagated ([0 0 1]) plane wave.



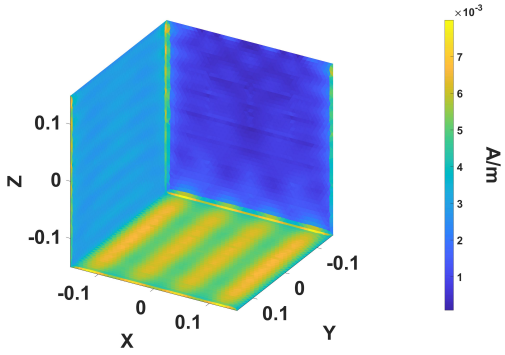
(a) Scattered E-fields from 1λ cube geometry.



(b) Scattered E-fields from 4λ cube geometry.



(c) Induced currents on 1λ cube.



(d) Induced currents on 4λ cube.

Figure 4.4: The comparison of scattered fields between MoM and MLFMA solutions ((a),(b)) and induced currents ((c),(d)) for different electrical sizes of cubes used for validation analyses.

4.2 MLFMA GPU Implemention Results

In this section, MLFMA GPU implementation results are presented. The first set of results are preliminary results for different implementation approaches. They show that the proposed approaches are valid and worthy of trying on different problems to assess their efficiency. Secondly, the actual implementation results are given for three different geometries, namely a PEC sphere (see Figure 4.1), a PEC aircraft geometry (see Figure 4.21) and a PEC missile geometry (see Figure 4.27). All real-life geometries (the aircraft and the missile) are found from open sources.

Two different hardware systems are used for testing the GPU implementations. The first system is a personal computer (PC). Its CPU is an Intel i9 2.40 GHz processor supported by 64 GB RAM, and its GPU is an NVIDIA GTX 1650ti with 896 cores supported by 4 GB global memory. The second system is a server computer. Its CPU is an Intel Xenon Gold 6248R 3.00 GHz processor supported by 567 GB RAM, and its GPU is an NVIDIA Quadro RTX 8000 graphics card with 4608 cores supported by 48 GB global memory. For the PEC sphere geometry, analyses are repeated on both hardware systems, whereas for real-life geometries, analyses are done only in the server system.

4.2.1 Preliminary Analyses for Testing Different GPU Implementation Approaches

As discussed in Chapter 3, there can be two different approaches (see Section 3.3.2 at page 45) when determining the size of each partition in our proposed matrix partition scheme. This part presents the preliminary test results of both approaches on different parts of MLFMA to see their performance outputs. The preliminary results are implemented on PC.

For the first approach, MLFMA is tested for different subroutines, namely Z_{near},

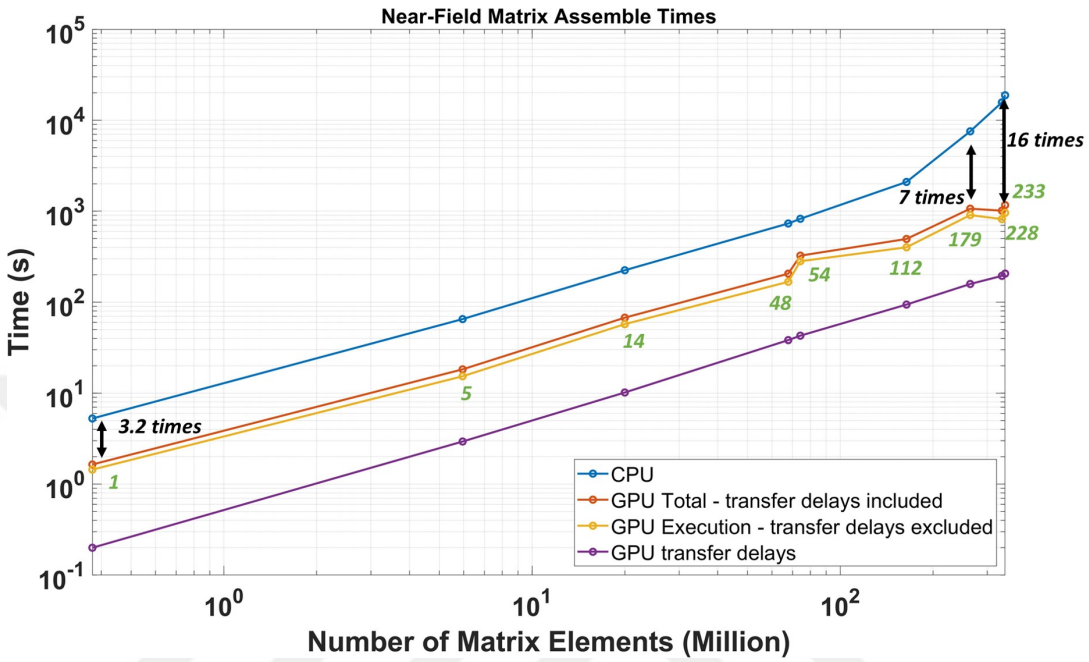


Figure 4.5: Near-field timings for the first approach of GPU implementation solved on PC

radiation/receiving patterns, interpolation & shifting, and translation. Maximum package size is calculated for each subroutine depending on the calculation algorithm in the previous chapter. For this purpose, different electrical sizes (from 1λ to 48λ) of sphere geometries are used with corresponding operating frequencies (from 1GHz to 48GHz).

In Figure 4.5, the near-field timings for the first approach of our GPU implementation are compared with the single-core CPU results. The x-axis of the graph is the number of near-field interactions (i.e., the total number of elements in the sparse Z_{near} column matrix, see Figure 3.3(a)), and the y-axis is the timings in seconds. In order to see the impact of the data transfer delays on the overall efficiency of the execution; the execution time, the GPU execution time (transfer delays excluded), and the transfer delays are plotted in Figure 4.5. The green numbers on the plot represent the total number of cycles required for our proposed scheme. It is observed that, as the number of matrix elements increases, the total cycles are also increasing since the total memory requirement increases depending on the size of the Z_{near} matrix. We can conclude that significant time efficiencies

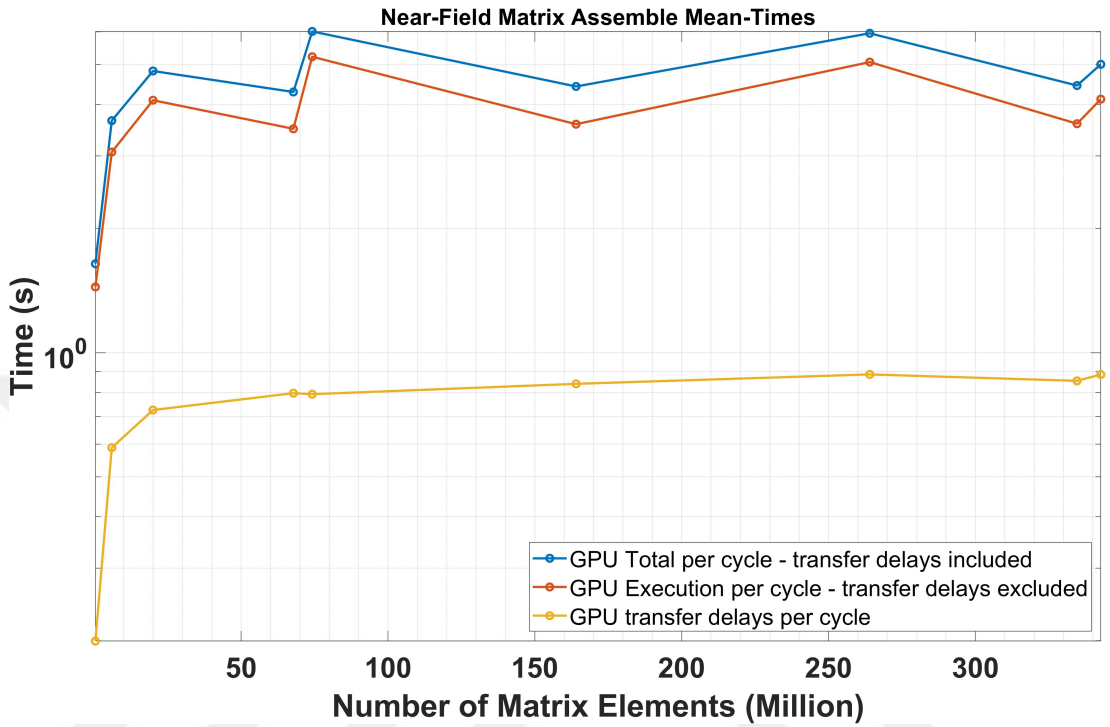


Figure 4.6: Near-field average timings for the first approach of GPU implementation solved on PC

(∼3 times to ∼16 times, illustrated with black arrows in Figure 4.5) are obtained when our proposed GPU implementation is employed for the calculation of Z_{near} matrix.

In Figure 4.6, the mean timings per cycle of the same GPU implementation (calculation of Z_{near} in Figure 4.5) are plotted. Since the size of the data transferred/processed at each cycle is the same (because the package size is determined at the beginning and it is not changed throughout the entire cyclic process), it is expected that the execution time per cycle and the transfer delay per cycle to be the same for all size of problems. Nevertheless, the timing values first show a sharp increase and then get stable in a region. This is because the first value in the graph belongs to the smallest problem (i.e., smallest Z_{near} matrix), and it requires only one cycle. Thus its timing value is not a good reference value because it takes time for GPU to stabilize its performance [38].

Furthermore, the reason behind the fluctuations for GPU execution times (which

is anticipated to be stable as a parallel line to the x-axis) is the performance variation of the GPU card during the process. This performance variation may result from some facts such as being affected by operating systems, the GPU usage of other applications, and also insufficient cooling capacities of laptops. This performance variation can be observed by the log-file of the analyses depicted in Figure 4.7, which shows the change of the execution times from ∼3.6 seconds to ∼11.7 seconds (see the red dashed box) for some period of the analyses. Even so, the general behavior of the GPU implementation is almost as expected, in other words the execution times and the data transfer delays stay in a value range, and this range shows the capacity of our GPU card for this specific application.

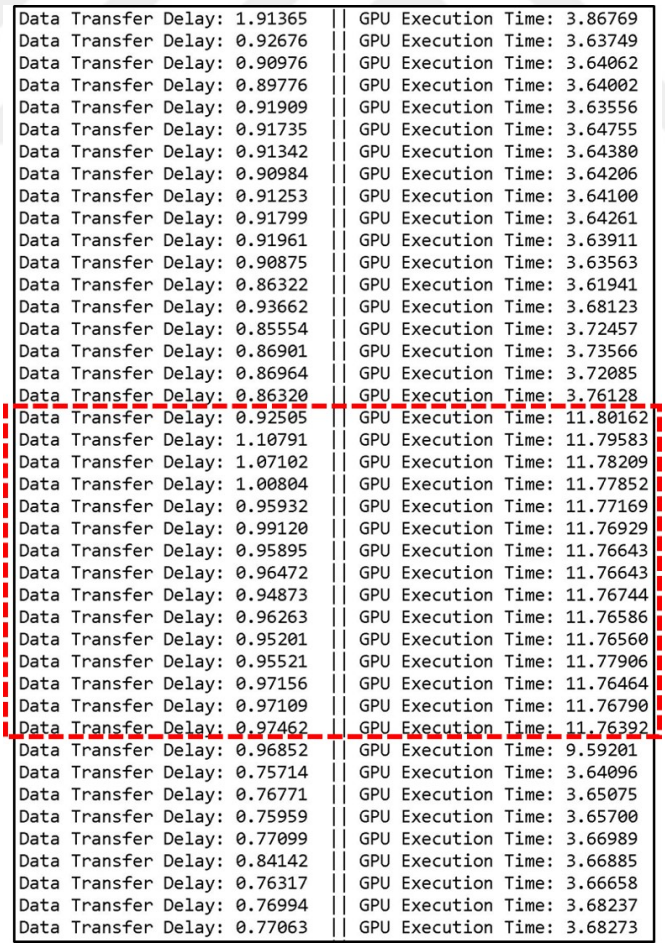


Figure 4.7: A part of the log-file of a GPU implementation for the calculation of near-field matrix solved on PC

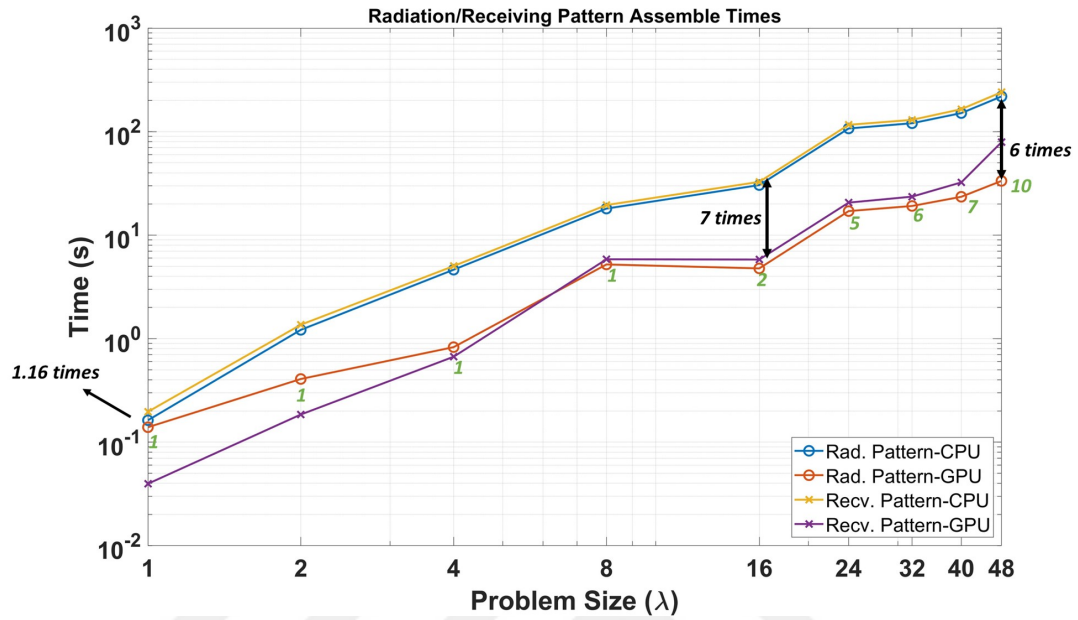


Figure 4.8: Radiation/receiving pattern timings for the first approach of GPU implementation solved on PC

In Figure 4.8, the radiation/receiving pattern timings of the first approach are compared with the single-core CPU timings. The x-axis of the graph is the electrical size of the problem (i.e., the physical size of the problem in terms of the wavelength of the illumination frequency), and the y-axis is the timings in seconds. The green numbers are the total cycles for each analysis, and they are relatively small numbers with respect to the same numbers in Figure 4.5, because the total size of the radiation/receiving patterns are relatively smaller than the near-field matrices of the same problems. The performance improvement increases ($\sim$1.16 times to $\sim$7 times) can be observed from the graph as shown with black arrows. Normally, it is expected for the radiation pattern calculation to be close to the receiving pattern calculation of GPU implementation for all values (i.e., red-line and purple-line, respectively). However, there is a deviation for the first two values (low-performance increase for 1λ and 2λ problems) because of the performance variation of GPU cards. Lastly, a good performance improvement (up to 7 times) is obtained.

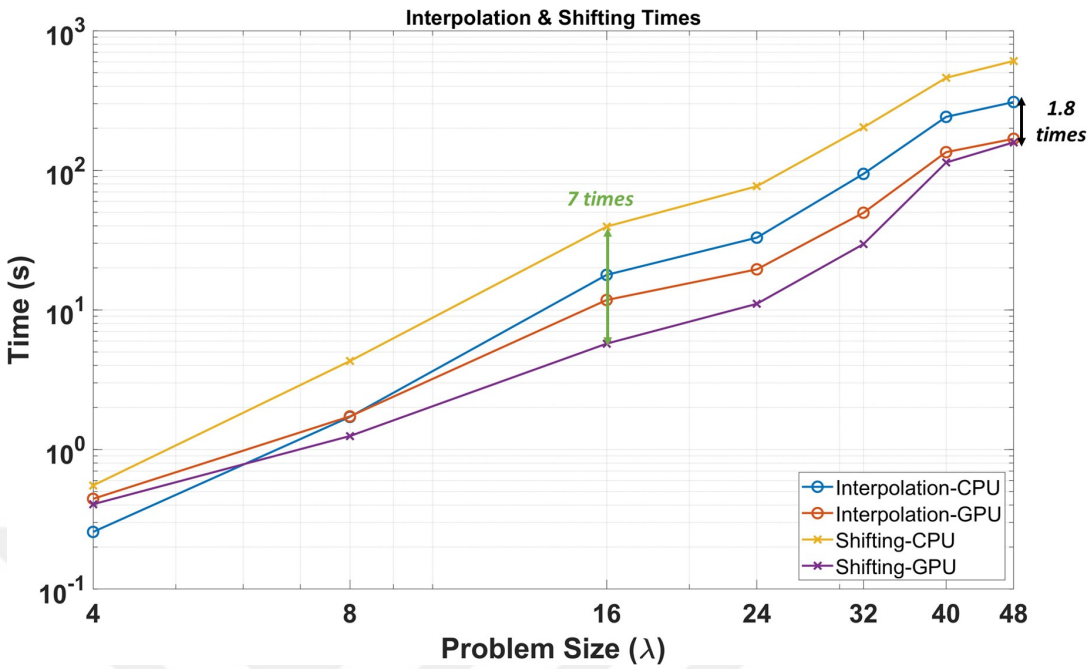


Figure 4.9: Interpolation & shifting timings for the first approach of GPU implementation solved on PC

In Figure 4.9, the timings of the interpolation process and shifting process of the first approach are compared with the single-core CPU timings. The x-axis of the graph is the electrical size of the problem, and the y-axis is the timings in seconds as before. In this graph, there are no cycle numbers because of two reasons: first, the total number of cycles required for interpolation & shifting processes are almost one for all problems; and second, since these processes occur at each level-switch during aggregation (i.e., at each sub-level aggregation from level L to L-1), there is no single cycle number for a problem but rather a cycle number for each level-switch (almost one for all level switch, at most two for some very high levels such as from 9 to 8). The performance improvement for the interpolation process is relatively small ($\sim$1.6 times at most, illustrated with black arrow), whereas the shifting process can be accelerated up to $\sim$7 times (illustrated with green arrow). This is because, in the interpolation process, in addition to the matrix-vector-multiplications, there are padding operations and also some other mathematical arrangements in our implementation. Therefore, the interpolation process is not a pure SIMD process that can explain its lower performance. Since the shifting process is an element-wise multiplication, it is a pure SIMD process, which provides higher performance improvement than the interpolation implementation.

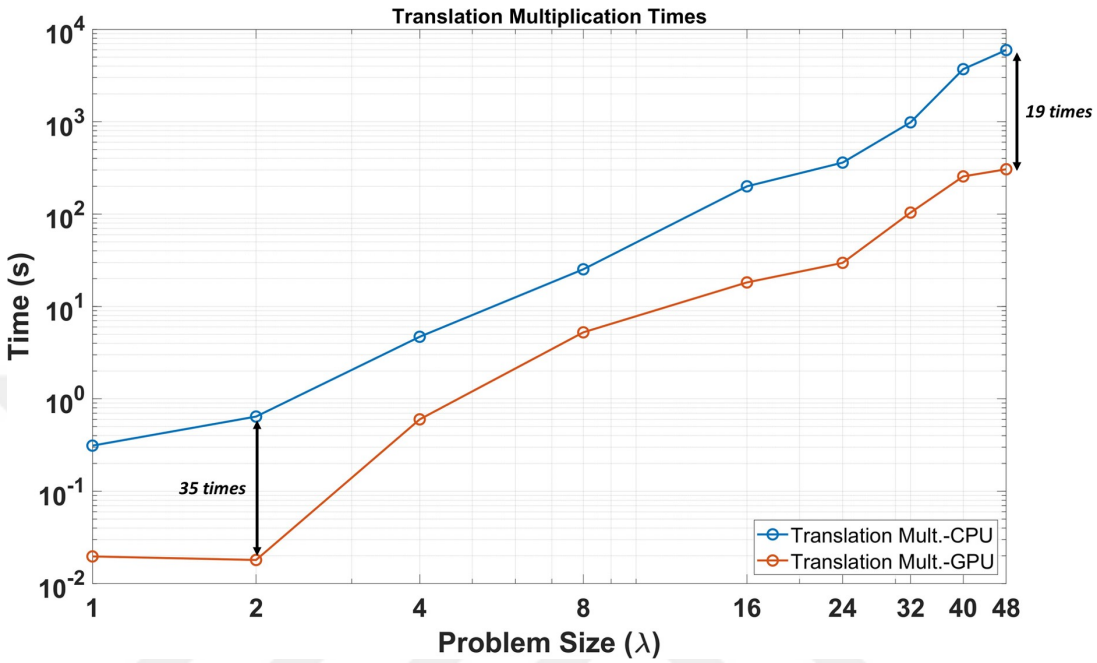


Figure 4.10: Timings of multiplication process of translation subroutine for the first approach of GPU implementation solved on PC

In Figure 4.10, the timings of multiplication process of the translation stage (i.e., element-wise multiplication in the translation stage, see Figure 3.4(c)) are presented. There are no cycle numbers illustrated on the graph because for each level of MLFMA, different number of cycle numbers are present. For instance, for the 32λ problem, there are eight levels in total, and the cycle numbers are 2, 3, 3, 5, 7, 12 for levels 3, 4, 5, 6, 7, 8, respectively. The multiplication process of the translation stage is a weighted process, which means that the total number of element-wise multiplications is relatively higher than that of the shifting process, and this is why the performance enhancement is much higher than the shifting process.

To conclude, for the first approach, we obtain different efficiencies for different parts of MLFMA depending on the compatibility of their mathematical process with the SIMD structure. In addition, we observe that the amount of acceleration becomes better as the operational weight of the SIMD process is increased (i.e., the total number of independent units of works is increased). Finally, the performance variation of GPU cards has a notable effect on the performance

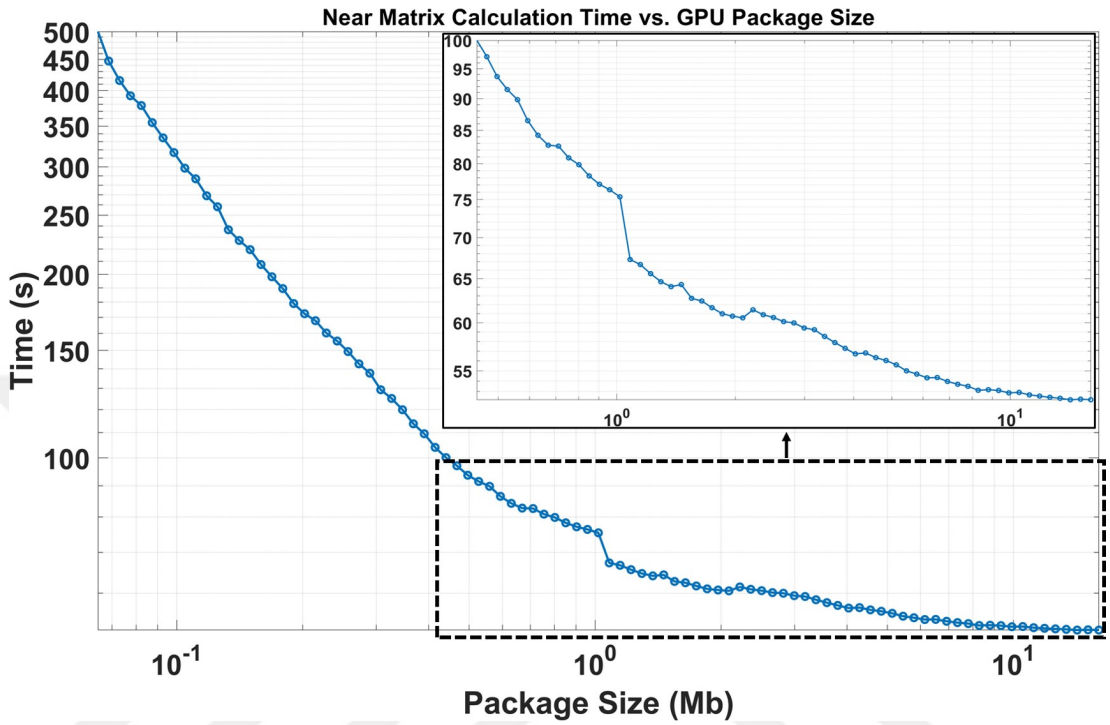
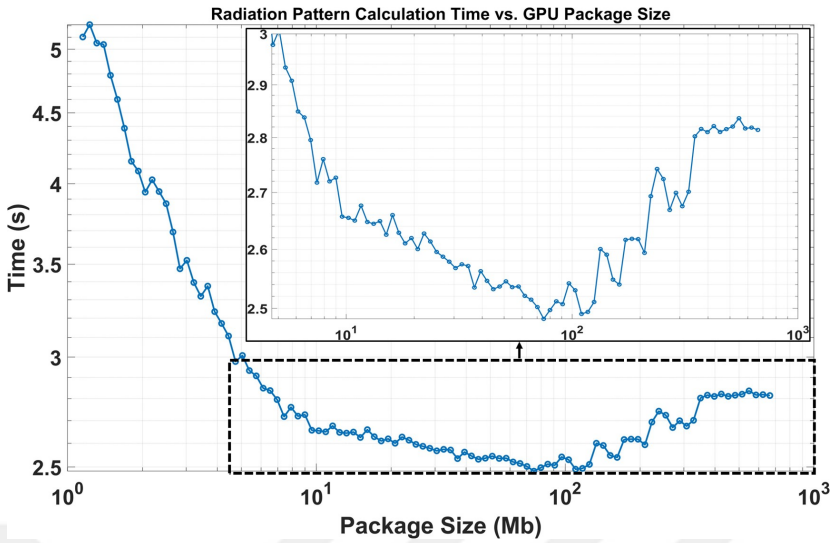


Figure 4.11: Near-Field timing test for the second approach of GPU implementation solved on PC

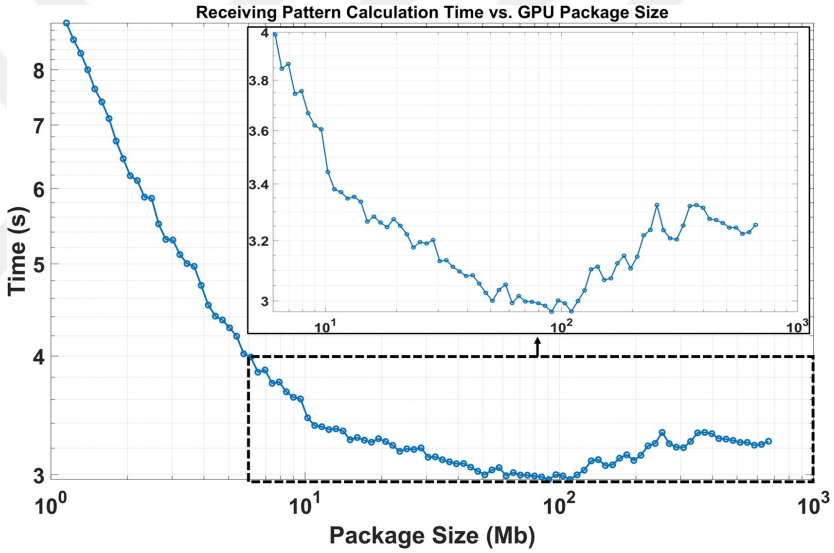
of the implementation.

For the second approach, MLFMA is tested for only the subroutines of near-field and radiation/receiving patterns because these two parts dominate the total timing of the entire analysis. For this purpose, different package sizes are calculated between the maximum and a minimum value for this purpose. Previously in the first approach, the maximum value was calculated for each subroutine. The minimum value for package sizes can be determined separately for each subroutine. For the near-field subroutine, single element of the Z_{near} matrix (see Figure 3.3(a)) can be chosen as the single partition (the minimum package), and for the radiation/receiving patterns, the single pattern of an RWG function (see Figure 3.3(c)) can be chosen as the single partition.

In Figure 4.11, x-axis is the package size of single partition and y-axis is the timing value of the near-field calculation. In other words, the same Z_{near} matrix is calculated repeatedly by using different package sizes for our implementation.



(a)



(b)

Figure 4.12: Radiation/Receiving pattern timing test for the second approach of GPU implementation solved on PC

Although the calculated subroutine is the same for all calculations, resulting timings vary from ∼500 seconds to ∼50 seconds as the package size increases. Since the package size decreases, the total number of cycles increases, and hence the device/memory initialization effects (occurs at each data transmission, [29, 38]) decreases the overall performance, and the total timing increases. The minimum timing value is obtained when the maximum package size is used. To conclude, we experience that the first approach (i.e., maximizing the package size) is more efficient for the calculation of Z_{near} matrix for our proposed GPU implementation.

In Figure 4.12, x-axis is the package size of single partition and y-axis is the timing value of radiation (4.12(a)) and receiving (4.12(b)) pattern calculations. The same procedure is followed as before, and the timing values are compared for different calculations of different package sizes. For this part, the timing values decrease to some point, (see Figure 4.12) before reaching the maximum package size. Then, after a switch-spot, it increases up to some value, as observed in the magnified parts of the graphs. The acquired efficiency of the minimum values (i.e., the absolute minimum of the graphs) concerning the value occurring at the maximum package size is around 10%. Since the operational weight of radiation/receiving patterns is much lower than Z_{near} matrix, the effects of internal memory delays may result in this type of behavior for our implementation, and a balance point can be observed. So, these results show that for the calculation of radiation/receiving patterns, the proposed GPU implementation can be tuned with optimum package size.

To sum up, the preliminary analyses of our proposed algorithm are tested, and resulting timings are obtained. It is observed that our proposed scheme is working, and satisfactory time efficiencies are obtained for each part of MLFMA. In addition, optimization tests are also done to enhance our proposed scheme, and using the second approach, the time efficiencies can be further increased. In the following, the results of GPU implementation for various geometries are presented.

4.2.2 Sphere Results Solved on PC

This part presents the results of MLFMA GPU implementation done in PC. The timings of GPU implementation are compared with timings of single-core CPU implementation. Nine different sphere geometries are used to show the efficiency of our GPU implementation. Their electrical sizes vary from 1λ to 48λ, and operating frequencies vary from 1GHz to 48GHz proportional to the electrical size (i.e., 1λ at 1GHz, 2λ at 2GHz, etc.). The number of unknowns (i.e., number of RWG functions) varies from 1281 to 2566713. The detailed specifications of the problems are shown in Table 4.1.

All of the spheres are illuminated by a linearly polarized plane-wave. Plane-wave is unit magnitude, x-polarized ([1,0,0]) and propagating along the $+z$-direction ([0,0,1]) as shown in Figure 4.1. The resulting scattered fields are calculated along the $\phi = 0^\circ$ cut (xz-plane), where θ values vary from 0 to 360 degrees with 1 degree spacing (see the red circle shown in Figure 4.1). In the numerical integrations over the triangles, 1-point testing function and 3-point basis function are used for the near-field interactions, and 3-point testing function and 3-point basis function are used for the far-field interactions.

The resulting scattered fields of some analyses and their induced currents are shown in Figure 4.13 and Figure 4.14, respectively. The complete result set of sphere analyses solved on PC can be found in Appendix A.1 for scattered fields and in Appendix B.1 for induced currents.

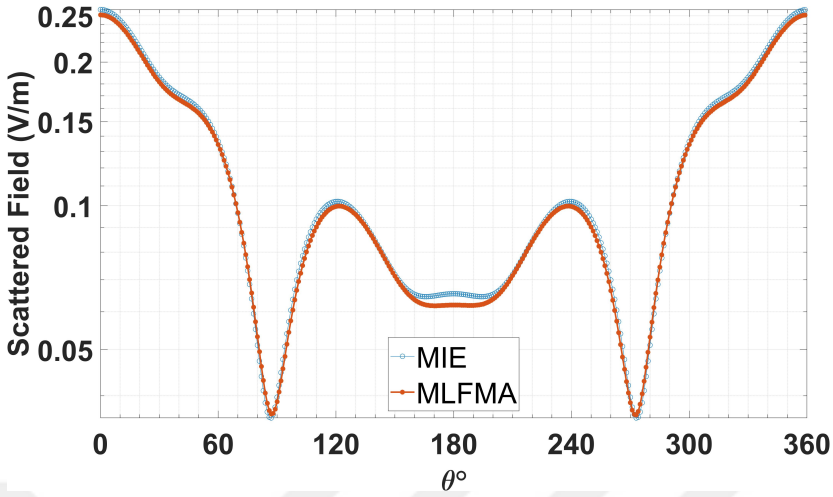
The maximum norm error is observed as 2.42%, and the minimum norm error is observed as 0.30%; thus, the results are quite accurate, and this can also be observed from the comparison graphs of scattered fields with Mie series solutions in Figure 4.13. The number of iterations (required for the convergence of each problem to the tolerance value of 10^{-3}) starts from 24 and increases to 44 as the electrical size of the problem increases. The largest problem size that can be reached in the PC is 48λ with 2566713 unknowns, and beyond this problem, RAM capacity prevents us from going further. For the smallest problem, 3 level

MLFMA is used whereas for the largest problem 9 level MLFMA is used.

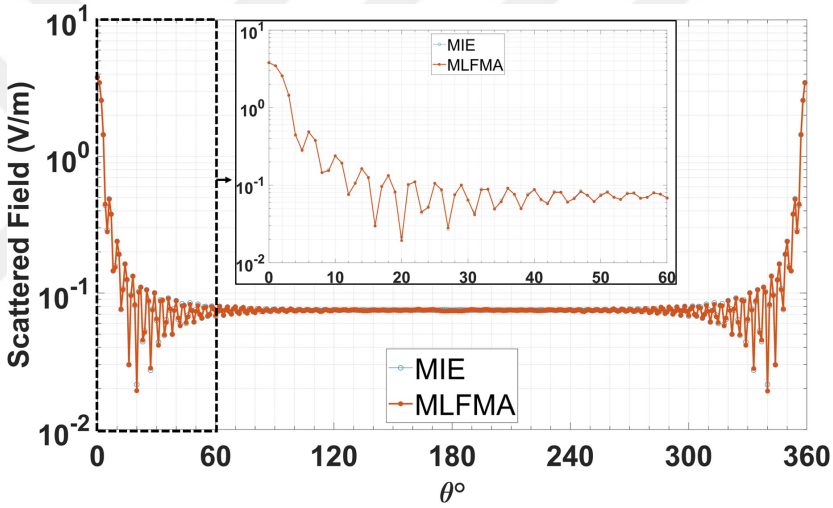
After many trials, the possible lowest MLFMA level is used for each problem to increase the overall efficiency of GPU implementation because the performance enhancement of near-field matrix is observed much higher than the other stages of MLFMA in the preliminary results. The relation between the size of near-field matrix and MLFMA level is explained as follows: The number of near-field interactions varies from 374245 to 342710896, and note that these values are not monotonically increasing (as the problem size increases) since the near-field matrix is dependent on multiple parameters such as MLFMA level, number of unknowns, and shape of the geometry. When the electrical size of the problem increases, then the number of unknowns and the size of near-field matrix increases as well, as the MLFMA level stays same. On the contrary, as the electrical size increases, MLFMA level needs to be increased as well at some point as the near-field matrix increases too much to be able to fit on RAM size. Thus, MLFMA level is also increased after some point as the problem size increases and at that level increase, the size of near-field matrix decreases. It can be observed from the Table 4.1 that switching from 24λ problem to 32λ, number of near-field interactions decreases (also for switching from 32λ problem to 40λ).

Electrical Size (λ)	Frequency (GHz)	RWG	MLFMA Level	NF Inter.	Iter.	Norm Error (%)
1	1	1281	3	374245	24	2.42
2	2	5100	3	5950426	26	1.13
4	4	19560	4	19989148	28	0.55
8	8	75297	5	74194057	31	0.30
16	16	288489	7	67791381	34	0.36
24	24	648948	7	342710896	37	0.37
32	32	1140126	8	264033504	39	0.35
40	40	1796160	9	164038728	42	0.59
48	48	2566713	9	334611187	44	0.55

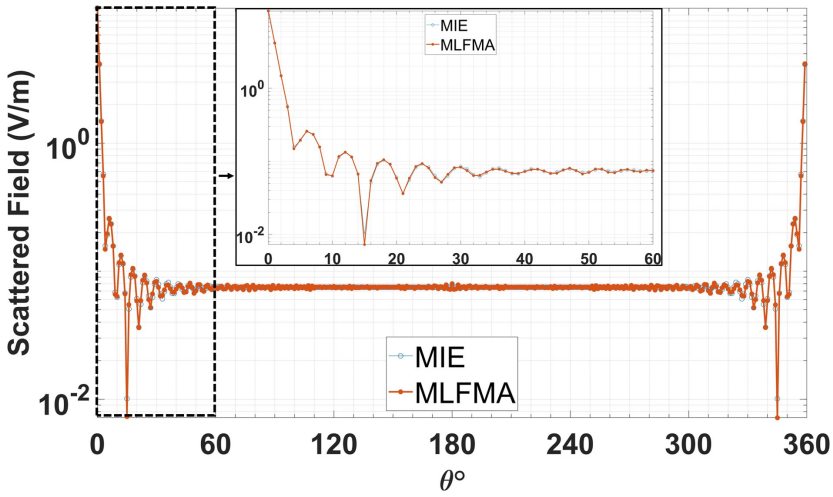
Table 4.1: Specifications of sphere problems solved on PC



(a) 1λ sphere

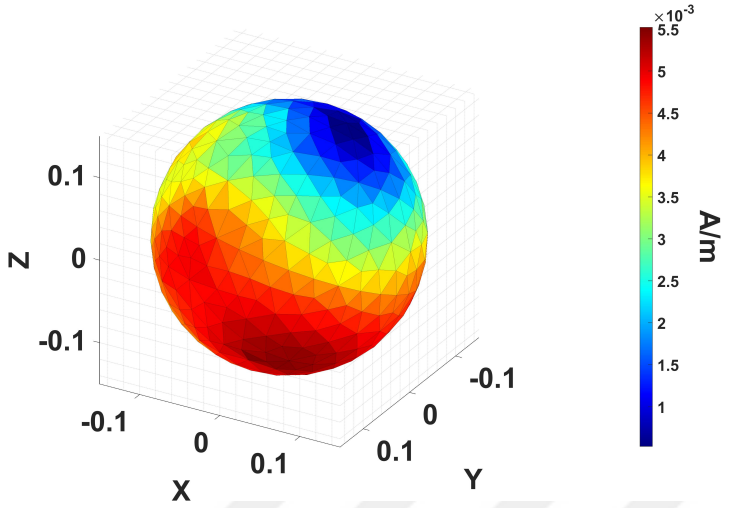


(b) 16λ sphere

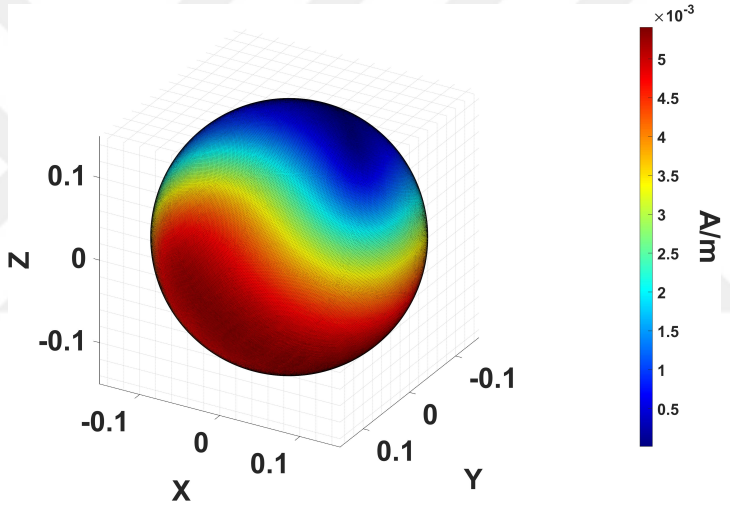


(c) 48λ sphere

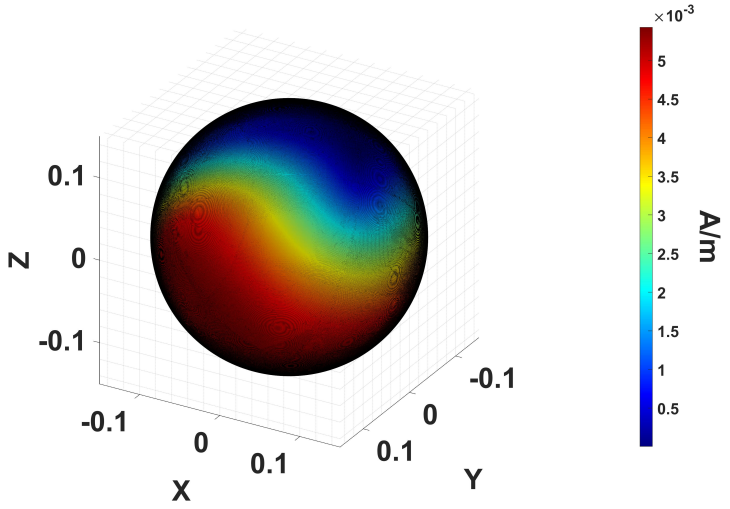
Figure 4.13: The resulting scattered fields compared with Mie series solution for the sphere analyses of $1\lambda, 16\lambda, 48\lambda$ spheres solved on PC



(a) 1λ sphere



(b) 16λ sphere



(c) 48λ sphere

Figure 4.14: The resulting induced currents for the sphere analyses of $1\lambda, 16\lambda, 48\lambda$ spheres solved on PC

	Hardware	Z_{near}	Lowest Level Agg.	Sub-Level Agg.	Trans.	Sub-Level DisAgg	Lowest Level DisAgg.	Total MVM	Total
1λ	CPU	5.27	4.70	1.05	0.89	0.63	5.26	12.52	17.79
	GPU	1.64	1.08	0.85	1.48	0.31	1.56	5.28	6.92
	Speed-up	3.21	4.34	1.23	0.61	2.01	3.38	2.37	2.57
2λ	CPU	65.16	34.47	1.18	1.20	0.75	36.54	74.14	139.30
	GPU	18.27	5.12	0.88	1.59	0.41	5.99	13.99	32.26
	Speed-up	3.57	6.73	1.33	0.76	1.85	6.10	5.30	4.32
4λ	CPU	224.42	140.29	1.82	6.65	1.52	150.26	300.55	524.96
	GPU	67.50	23.16	2.08	5.81	1.05	30.15	62.25	129.74
	Speed-up	3.32	6.06	0.87	1.15	1.45	4.98	4.83	4.05
8λ	CPU	825.25	598.16	7.61	34.51	7.51	637.55	1285.33	2110.59
	GPU	324.73	142.17	4.76	20.31	3.81	158.22	329.27	654.00
	Speed-up	2.54	4.21	1.60	1.70	1.97	4.03	3.90	3.23
16λ	CPU	732.88	1118.27	66.36	304.00	67.78	1181.95	2738.35	3471.23
	GPU	205.85	204.88	27.80	172.56	27.09	251.65	683.98	889.83
	Speed-up	3.56	5.46	2.39	1.76	2.50	4.70	4.00	3.90
24λ	CPU	18823.52	4292.21	126.76	558.65	126.18	4545.61	9649.43	28472.96
	GPU	1166.09	924.91	49.54	212.03	45.82	954.49	2186.81	3352.90
	Speed-up	16.14	4.64	2.55	2.63	2.75	4.76	4.41	8.49
32λ	CPU	7556.50	5009.69	343.46	1528.84	341.58	5284.88	12508.45	20064.95
	GPU	1065.51	1107.92	121.35	801.63	117.73	1171.07	3319.70	4385.21
	Speed-up	7.09	4.52	2.83	1.91	2.90	4.51	3.77	4.58
40λ	CPU	2098.46	6733.10	818.77	5227.38	905.42	7162.64	20847.31	22945.77
	GPU	495.32	1488.89	363.42	2878.50	339.07	1495.02	6564.90	7060.22
	Speed-up	4.24	4.52	2.25	1.82	2.67	4.79	3.18	3.25
48λ	CPU	15720.98	10010.10	1064.81	7904.56	1206.69	11345.73	31531.89	47252.87
	GPU	1013.70	2271.82	470.22	3111.78	420.00	2230.90	8504.73	9518.43
	Speed-up	15.51	4.41	2.26	2.54	2.87	5.09	3.71	4.96

Table 4.2: Time comparison of CPU and GPU implementations for different sphere geometries solved on PC.

In Table 4.2, detailed timings of the sphere analyses are given. For each part of MLFMA, timings are presented, namely the Z_{near}, lowest-level aggregation (aggregation denoted as agg. in short), sub-level aggregation, translation, sub-level disaggregation (disaggregation denoted as disagg. in short), lowest-level disaggregation, total MVM and total. The Total MVM indicates the sum of the lowest-level aggregation, sub-level aggregation, translation, sub-level disaggregation, and the lowest-level disaggregation, whereas the Total indicates the sum of the Total MVM and Z_{near}. For each analysis; CPU timing, GPU timing, and speed-up value (CPU/GPU) are given, and the left-most column indicates the electrical size of each problem.

Several conclusions can be done from the timing results presented in Table 4.2. We obtain 2.57 to 8.49 times performance enhancement with our GPU implementation in terms of total analysis time. Especially for large-scale problems like 48λ, the total time saved by GPU is almost 10.5 hours which is a significant saving, especially for practical applications.

For different parts of MLFMA, different speed-up values are obtained. One of the best acceleration is acquired for Z_{near} subroutine, which is up to 16 times. This is the expected conclusion (also observed in the preliminary results in Figure 4.5) since the assembling of Z_{near} matrix is pure SIMD (one of the most compatible subroutine with GPU structure) and computationally intensive (high operational weight per transferred byte) [29, 38, 57]. The lowest-level aggregation/disaggregation parts are another highly accelerated (up to 6.7 times) parts of MLFMA (sometimes more efficient than Z_{near} calculation) because of the same reasoning as in Z_{near} subroutine. The similar performance domination of Z_{near} and the lowest-level aggregation/disaggregation are observed in the literature as well [27–29].

The translation stage exhibits the lowest performance enhancement in MLFMA. Although the multiplication process (illustrated as the first process in Figure 3.4(c)) in the translation stage is an SIMD process and also gives very high speed-up values (see the preliminary results, Figure 4.10), the distribution process

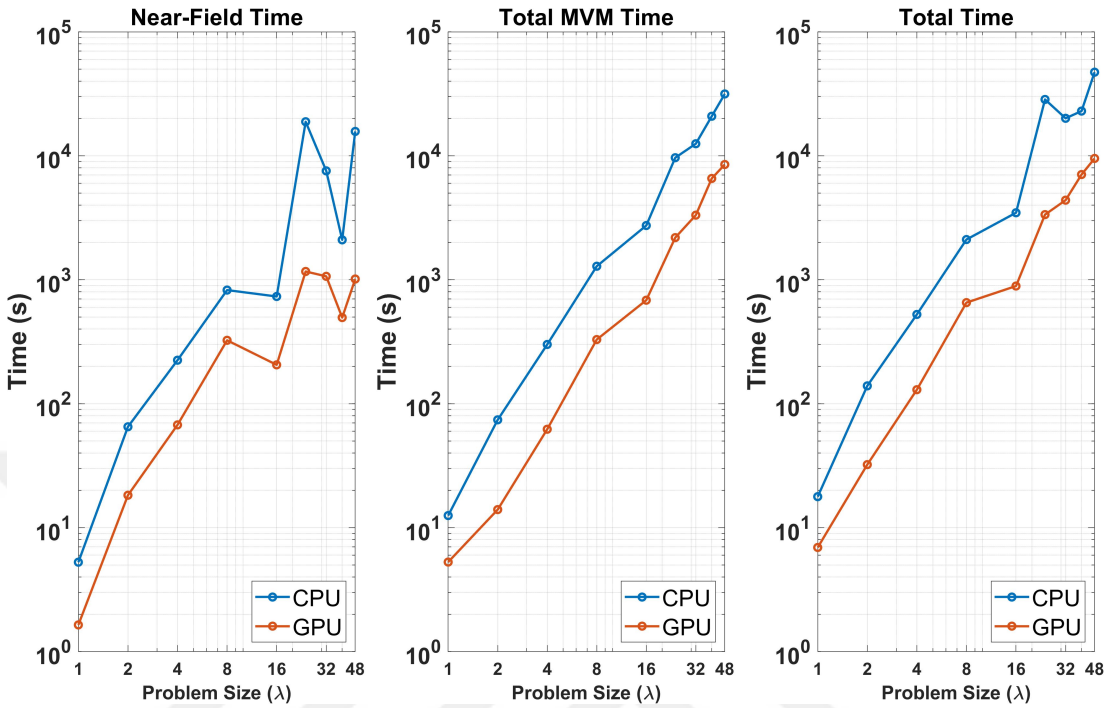


Figure 4.15: Near-Field and MVM and total times for sphere analyses solved on PC

(illustrated as the second process Figure 3.4(c)) of the translated fields to boxes decreases the overall performance enhancement remarkably. This is because the distribution stage is not an SIMD process and requires high memory access to distribute the fields to different boxes and also dominates the translation stage.

The sub-level aggregation/disaggregation parts have modest performance improvements in the overall execution. Although the acceleration achievement of the shifting process is good (see Figure 4.9), because of the low acceleration of the interpolation process (discussed in the preliminary results), the resulting performance enhancement of the sub-level aggregation/disaggregation stage stays in between the translation stage and the lowest-level aggregation/disaggregation stages.

The graphical representations of the values provided in Table 4.2 are illustrated in Figure 4.15 and Figure 4.16. In Figure 4.15; near-field timings, MVM timings and total timings are compared for CPU and GPU. The x-axis is the electrical size of the problems, and the y-axis is the timing values in seconds for both figures. As the problem size increases, the timing values also increase for MVM and the total

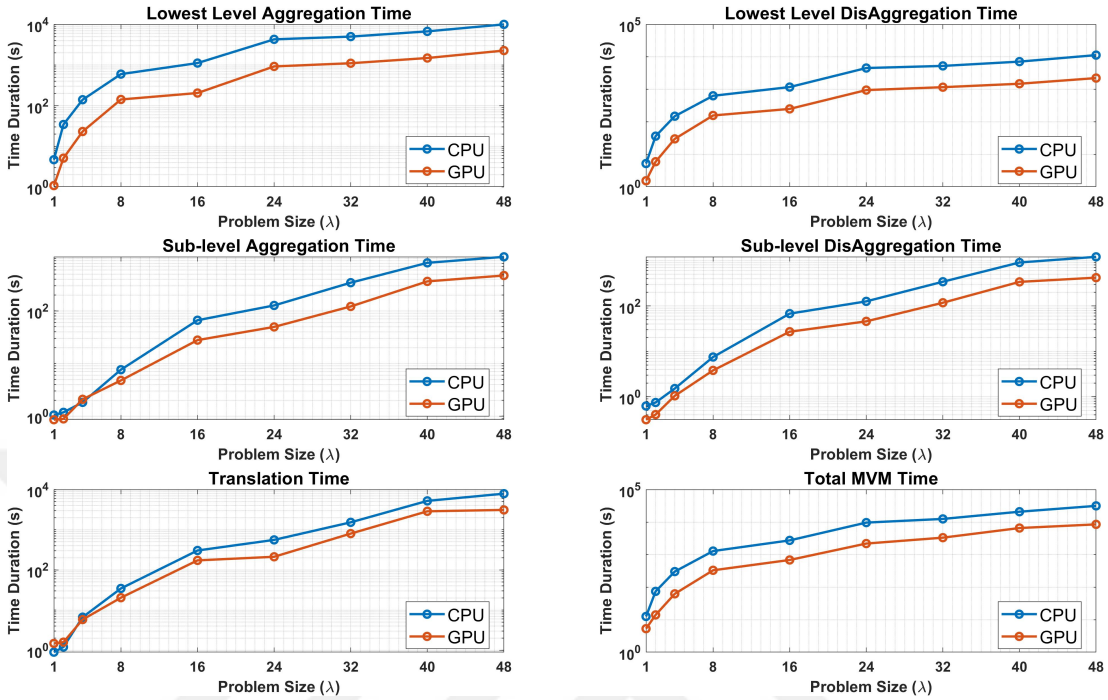


Figure 4.16: Details of MVM timing values for sphere analyses solved on PC

timing, whereas it fluctuates for the near-field timing. This is because the number of elements in Z_{near} matrix does not monotonically increase as discussed in Section 4.2.2 at page 65. Moreover, it can be observed in the total timing that a little peak occurs at 24λ problem, and this can be explained as the performance variation of CPU (performance degradation in this case) and GPU as discussed in the preliminary results. In Figure 4.16, timing values of each sub-stage of MVM stage are presented. Firstly, notice that the maximum performance enhancement can be clearly seen in the lowest-level aggregation/disaggregation stages. Secondly, the modest performance enhancement of sub-level aggregation/disaggregation stages can be observed. Lastly, the lowest performance enhancement of the translation stage is shown. In addition, in the translation stage, GPU timing results in performance degradation for the first two analyses. This can be explained by the domination of the distribution process being more than the multiplication process for the lower levels of MLFMA. Lastly, performance degradation is also observed in the sub-level aggregation for the 4λ sphere, which probably results from the performance variation (as discussed in Section 4.2.1 at page 57) of the GPU (see Figure 4.7) because there is no reasoning that we can foresight for this result. Lastly, the reason behind choosing the MLFMA level as low as possible is

because as the level decreases, the domination of near-field and the lowest-level aggregation/disaggregation calculation increase which are the most accelerated parts of MLFMA in our implementation.

In the literature, regarding the similar GPU MLFMA implementations on single-GPU systems with CFIE formulation [29, 30], there are very different speed-up values depending on the hardware systems and the problem sizes. For instance, $\sim$5.8 and $\sim$7.7 speed-up values are reported for total timing of MLFMA in [30] for problems with 60924 and 40725 unknowns, respectively, whereas $\sim$85 to $\sim$190 speed-up values are reported in [29] for problems with 19693, 85373, 694142 unknowns. Thus, making a fair comparison for the speed-up values is not possible because of different hardware systems (CPU and GPU specifications can vary considerably). Another reasoning for the impropriety of the speed-up comparison is that none of the similar works use double-precision arithmetic, and it is reported in the literature that GPU performances differ by precision arithmetic remarkably [55,56]. Furthermore, there are also some other implementations that use multi-GPU systems [27] or GPU-clusters [34,35]. Multi-GPU systems or GPU clusters are not in the scope of this work. However, to have an idea for future research directions on this topic, we can note that these publications report 5-20 times and 25-45 times speed-up values, respectively.

Another issue is the problem scales in the available publications. For example, the largest problem (in terms of unknowns) that is presented in [30] is (60924 unknowns) lower than our work (2566713 unknowns). An important point is also comparing the RAM capacities to make a fair comparison of problem scales. So, the GPU system used in [30] has a 256 MB RAM size which is almost 12 times lower than our GPU RAM (4 GB). However, the 12 times RAM difference does not compensate for the difference in the number of unknowns. For this purpose, taking the ratio of the number of unknowns per MB (i.e., megabyte value of GPU RAM) can be useful as a reference, and this value is 238.98, and 626.63 for [30] and our work, respectively. Moreover, in [29], the reported largest problem is 694142 with 6 GB GPU RAM; hence, the unknown per byte ratio is 115.69, which is relatively lower than our value. To the best of our knowledge, an important

remark is that we can solve large-scale problems up to 2566713 unknowns, whereas the highest reported value in the literature is 694142 unknowns on single-GPU systems [29].

4.2.3 Sphere Results Solved on Server

This part presents the results of MLFMA GPU implementation done in the server. The timings of GPU implementation are compared with single-thread CPU timings again. This time, 14 different sphere geometries are used to show the efficiency of our GPU implementation. Their sizes vary from 1λ to 88λ, and operating frequencies vary from 1GHz to 88GHz proportional to the electrical size. The number of RWG functions varies from 1281 to 8577765. The detailed specifications of the problems are shown in Table 4.3.

All of the spheres are illuminated by a a linearly polarized plane-wave. Plane-wave is unit magnitude, x-polarized ([1,0,0]) and propagating along the $+z$-direction ([0,0,1]) as shown in Figure 4.1. The resulting scattered fields are calculated along the $\phi = 0^\circ$ cut (xz-plane), where θ values vary from 0 to 360 degrees with 1 degree spacing as shown in Figure 4.1. In the numerical integrations over the triangles, 3-point testing function and 4-point basis function are used for the near-field interactions, and 3-point testing function and 3-point basis function are used for the far-field interactions.

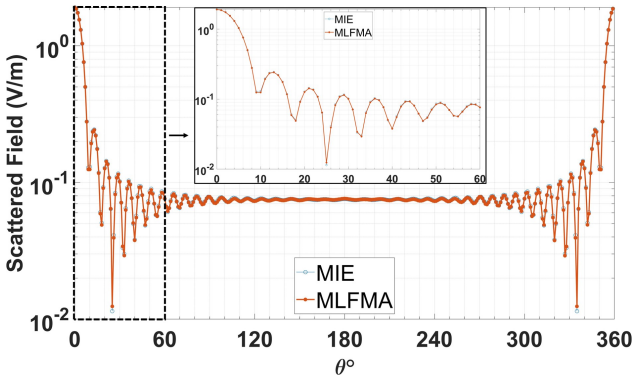
The resulting scattered fields of some analyses and their induced currents are shown in Figure 4.17 and Figure 4.18, respectively. The complete result set of sphere analyses solved on server can be found in Appendix A.2 for scattered fields and in Appendix B.2 for induced currents.

The maximum norm error is observed as 1.77%, and the minimum norm error is observed as 0.36%; thus, the results are quite accurate, and this can also be observed from the comparison graphs of scattered fields with Mie series solutions in Figure 4.13. The number of iterations starts from 19 and increases to 45 as

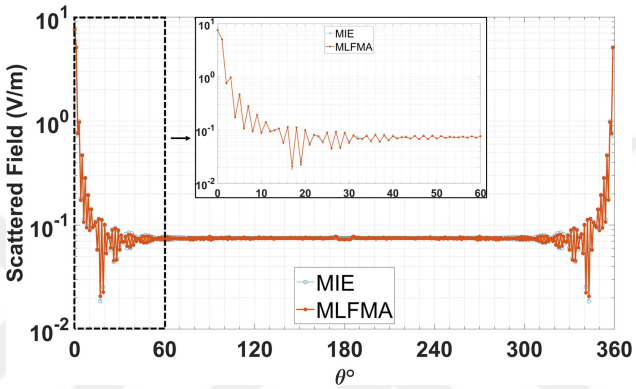
the electrical size of the problem increases. An important note is that the total iterations are lower for the sphere analyses solved in the server than the sphere analyses solved in PC (even the problems are identical for 1λ to 48λ) because the number of integration points on testing and basis functions are increased from 1-3 to 3-4, respectively, and this increases the near-field accuracy and thus decreases the required iterations. The largest problem size that can be solved in the server is 88λ with 8577765 unknowns, and beyond this problem, RAM capacity restricts us from going further. For the smallest problem, 3 level MLFMA is used; and for the largest one, 10 level MLFMA is used. Possible lowest MLFMA level is used like in the previous implementation. The number of near-field interactions varies from 374245 to 1048205623, and these values are not monotonically increasing, as explained in the Section 4.2.2 at page 65. It can be observed from the Table 4.3 that switching from 64λ problem to 72λ, number of near-field interactions decreases as we are changing the number of levels (also for switching from 24λ problem to 32λ and 32λ problem to 40λ).

Electrical Size (λ)	Frequency (GHz)	RWG	MLFMA Level	NF Inter.	Iter.	Norm Error (%)
1	1	1281	3	374245	19	1.52
2	2	5100	3	5950426	22	0.70
4	4	19560	4	19989148	23	0.41
8	8	75297	5	74194057	26	0.36
16	16	288489	7	67791381	29	0.43
24	24	648948	7	342710896	32	0.60
32	32	1140126	8	264033504	33	0.43
40	40	1796160	9	164038728	36	0.68
48	48	2566713	9	334611187	38	0.64
56	56	3474504	9	613210960	39	0.56
64	64	4543503	9	1048205623	40	0.64
72	72	5730237	10	417071713	42	1.77
80	80	7117743	10	643913015	44	0.90
88	88	8577765	10	934568741	45	0.71

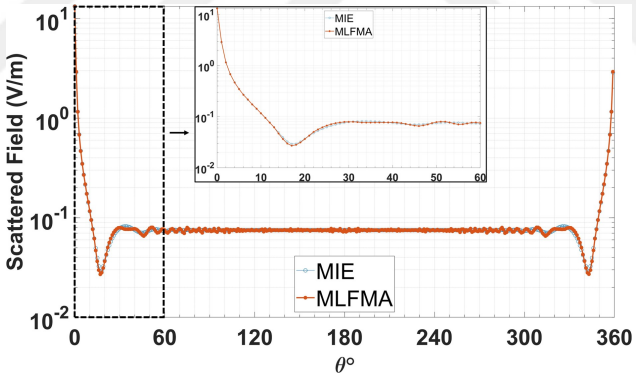
Table 4.3: Specifications of sphere problems solved on the server



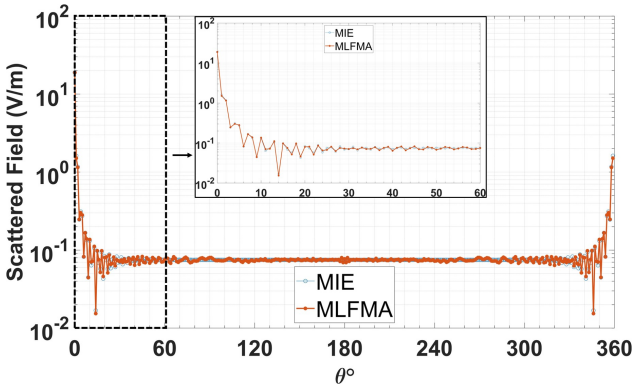
(a) 8λ sphere



(b) 32λ sphere

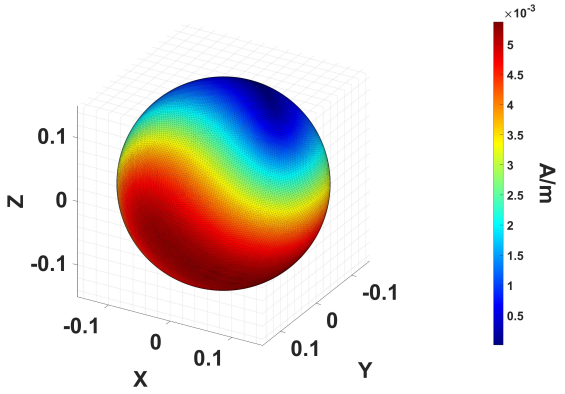


(c) 56λ sphere

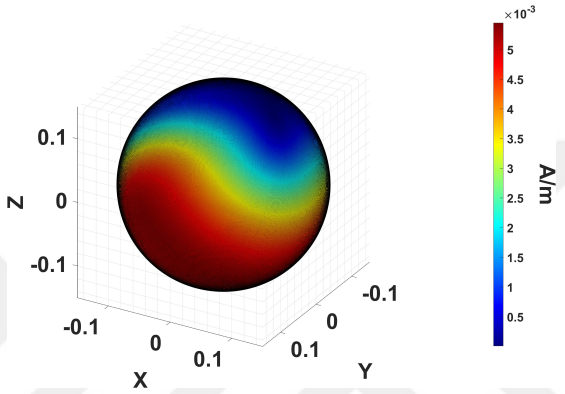


(d) 80λ sphere

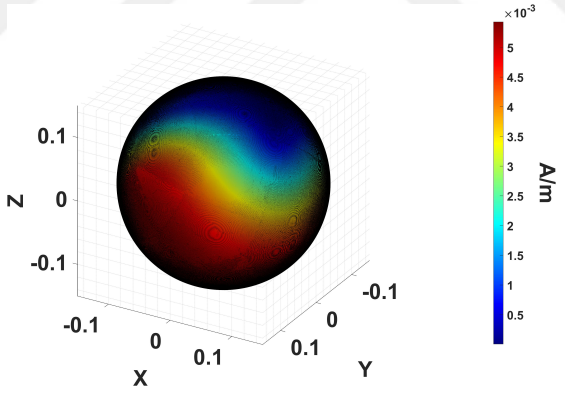
Figure 4.17: The resulting scattered fields compared with Mie series solution for the sphere analyses of $8\lambda, 32\lambda, 56\lambda, 80\lambda$ spheres solved on the server



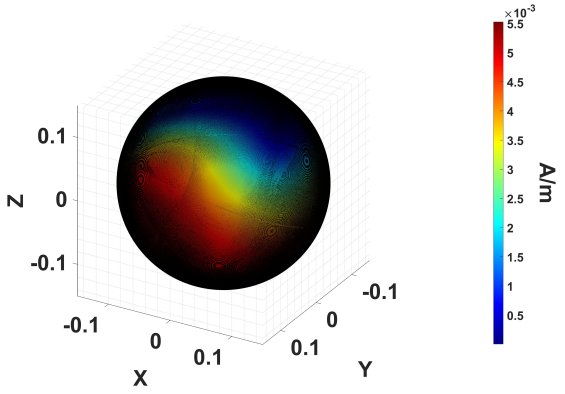
(a) 8λ sphere



(b) 32λ sphere



(c) 56λ sphere



(d) 80λ sphere

Figure 4.18: The resulting induced currents for the sphere analyses of $8\lambda, 32\lambda, 56\lambda, 80\lambda$ spheres solved on the server

	Hardware	Z_{near}	Lowest Level Agg.	Sub-Level Agg.	Trans.	Sub-Level DisAgg	Lowest Level DisAgg.	Total MVM	Total
1λ	CPU	13.76	3.10	0.81	0.68	0.35	3.36	8.31	22.07
	GPU	3.27	1.16	0.32	0.63	0.10	1.21	3.42	6.69
	Speed-up	4.21	2.67	2.56	1.08	3.47	2.78	2.43	3.30
2λ	CPU	337.68	30.72	0.56	0.99	0.26	32.72	65.25	402.93
	GPU	23.90	5.59	0.46	0.96	0.12	6.30	13.44	37.34
	Speed-up	14.13	5.49	1.20	1.03	2.11	5.19	4.85	10.79
4λ	CPU	1225.20	158.15	1.53	6.63	1.30	170.40	338.01	1563.21
	GPU	77.45	28.67	3.72	2.47	0.50	31.66	67.02	144.47
	Speed-up	15.82	5.52	0.41	2.69	2.60	5.38	5.04	10.82
8λ	CPU	4552.12	712.08	6.99	41.34	5.88	761.21	1527.50	6079.62
	GPU	289.00	121.04	3.58	8.16	1.88	132.53	267.19	556.19
	Speed-up	15.75	5.88	1.95	5.07	3.13	5.74	5.72	10.93
16λ	CPU	4101.83	1364.25	75.89	355.21	73.19	1440.94	3309.48	7411.30
	GPU	263.72	248.10	23.75	113.98	19.02	257.01	661.86	925.58
	Speed-up	15.55	5.50	3.20	3.12	3.85	5.61	5.00	8.01
24λ	CPU	22134.99	5132.15	150.56	637.53	145.08	5516.33	11581.66	33716.65
	GPU	1313.26	903.84	43.80	145.49	36.35	999.57	2129.05	3442.31
	Speed-up	16.86	5.68	3.44	4.38	3.99	5.52	5.44	9.79
32λ	CPU	16576.22	5970.86	404.66	1710.10	402.85	6307.87	14796.33	31372.55
	GPU	1019.44	1108.63	116.85	536.14	103.94	1144.29	3009.84	4029.28
	Speed-up	16.26	5.39	3.46	3.19	3.88	5.51	4.92	7.79
40λ	CPU	8966.17	7139.97	870.50	4069.52	952.36	7729.36	20761.72	29727.89
	GPU	640.28	1479.83	273.33	2148.01	297.10	1548.94	5747.22	6387.49
	Speed-up	14.00	4.82	3.18	1.89	3.21	4.99	3.61	4.65
48λ	CPU	18995.42	10731.13	1167.61	5018.27	1238.84	11297.49	29453.34	48448.76
	GPU	1317.11	2268.55	354.49	2262.63	359.60	2318.65	7563.93	8881.04
	Speed-up	14.42	4.73	3.29	2.22	3.45	4.87	3.89	5.46

Table 4.4: Time comparison of CPU and GPU for different sphere geometries solved on the server

	Hardware	Z_{near}	Lowest Level Agg.	Sub-Level Agg.	Trans.	Sub-Level DisAgg	Lowest Level DisAgg.	Total MVM	Total
56λ	CPU	40086.56	21655.88	1645.43	7420.18	1733.86	23080.20	55535.55	95622.11
	GPU	2386.22	4085.31	473.98	2601.15	482.77	4899.93	12543.13	14929.35
	Speed-up	16.80	5.30	3.47	2.85	3.59	4.71	4.43	6.40
64λ	CPU	75279.70	30072.65	2099.93	9438.38	2188.92	32407.59	76207.47	151487.17
	GPU	4042.10	5464.14	563.10	2654.01	566.54	6957.16	16204.95	20247.04
	Speed-up	18.62	5.50	3.73	3.56	3.86	4.66	4.70	7.48
72λ	CPU	26602.62	23877.02	4178.02	20998.32	4495.70	25194.05	78743.11	105345.73
	GPU	1621.40	4800.90	1272.25	10167.94	1280.74	5334.86	22856.68	24478.08
	Speed-up	16.41	4.97	3.28	2.07	3.51	4.72	3.45	4.30
80λ	CPU	44049.35	40998.58	5014.11	25763.72	5545.71	43533.74	120855.86	164905.21
	GPU	2610.39	9824.88	1390.10	10527.52	1514.78	9142.51	32399.80	35010.19
	Speed-up	16.87	4.17	3.61	2.45	3.66	4.76	3.73	4.71
88λ	CPU	67068.53	50702.67	6079.50	29832.78	6588.75	54208.38	147412.07	214480.60
	GPU	3599.70	11162.68	1645.83	10962.00	1679.67	10827.10	36277.27	39876.96
	Speed-up	18.63	4.54	3.69	2.72	3.92	5.01	4.06	5.38

Table 4.5: Time comparison of CPU and GPU for different sphere geometries solved on the server

In Table 4.4 and Table 4.5, detailed timings of the sphere analyses are given. For each part of MLFMA, timings are presented as before.

Several conclusion can be done from the timing results shown in Table 4.4 and Table 4.5. We obtain 3.3 to 10.93 times performance enhancement with our GPU implementation in total analysis time. Especially for large-scale problems like 88λ, the total time saved by GPU is almost 48.5 hours. Similar to the previous implementation, best accelerations are acquired for Z_{near} subroutine and the lowest-level aggregation/disaggregation stages, which are up to 18 times and 5.8 times, respectively. The translation stage shows the lowest performance, with

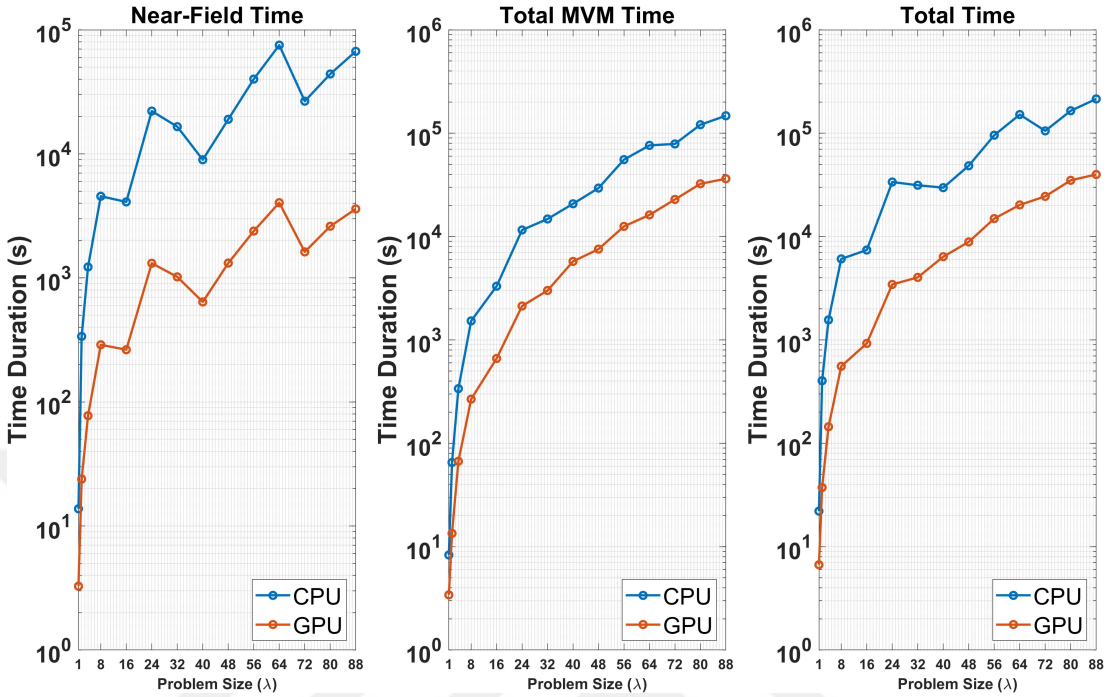


Figure 4.19: Near-Field and MVM and total times for sphere analyses solved on the server

speed-up values varying from 1.03 to 5. The sub-level aggregation/disaggregation parts exhibit modest performances enhancement with speed-up values vary from 0.41 to 3.92.

Figure 4.19 and Figure 4.20 are the graphical representations of Table 4.4 and Table 4.5. The x-axis is the electrical size of the problems and y-axis is the timing values in seconds for both figures. In Figure 4.19; near-field timings, MVM timings and total timings are compared for CPU and GPU. Compared with Figure 4.15, the unwanted peak at 24λ (results from the performance variation of CPU or GPU) is not seen in Figure 4.19. The reason behind it is probably the stability of the server systems, which are much better than that of a PC because of the Linux based operating systems and high-performance cooling infrastructures. The fluctuation occurred in Figure 4.19, is again because of the varying number of near-field interactions.

In Figure 4.16, timing values of each sub-stage of MVM stage were presented when the spheres were solved on PC. Similar remarks can be noted here in Figure 4.20 as well. Firstly, the maximum performance improvement can be

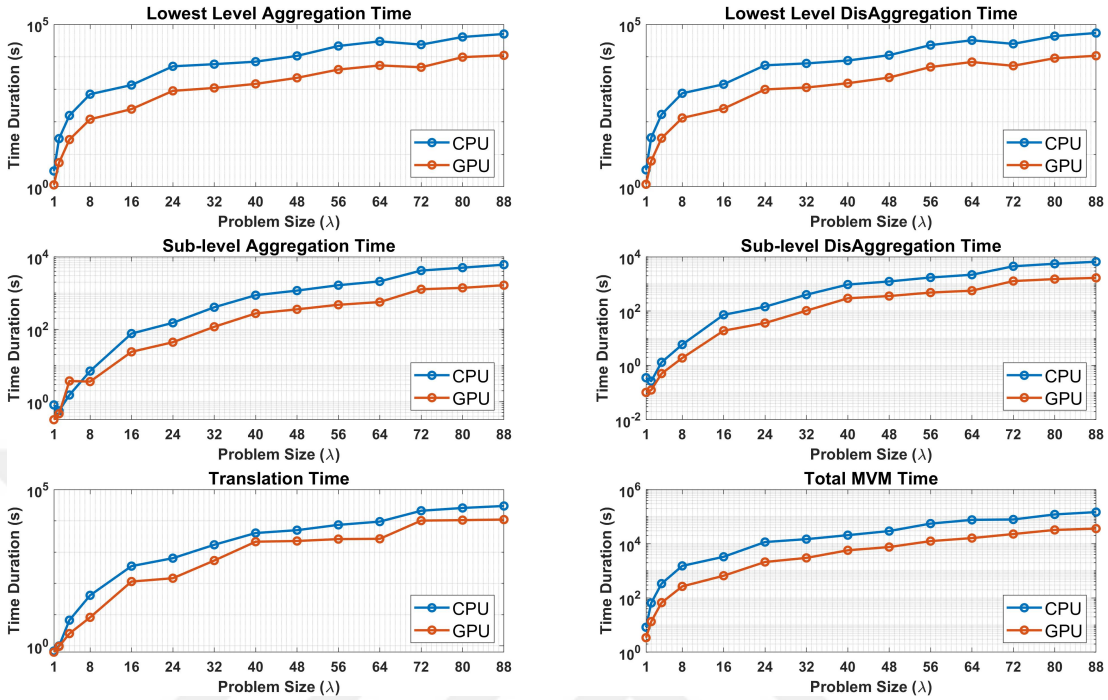


Figure 4.20: Details of MVM timings for sphere analyses solved on server

observed in the lowest-level aggregation/disaggregation stages. Secondly, the modest performance enhancement of sub-level aggregation/disaggregation stages is shown, and lastly, the lowest performance enhancement is presented for the translation stage. Moreover, in the translation stage, there is no performance degradation whereas there was in Figure 4.16 for the PC implementation. This can be explained by the higher processing capacity of the GPU card of the server than that of the GPU card of PCs. To conclude, the sphere results obtained in the server are more efficient and stable than that of the PC, and this can be because of the high-performance capacity, stable operating systems, and powerful cooling infrastructures of the server. In the rest of the numerical results chapter, real-life problems of MLFMA GPU implementation are given.

4.2.4 Aircraft Results

In this part, the results of MLFMA GPU implementation for an aircraft geometry (as shown in Figure 4.21) are presented. The results are obtained on the server system. The timings of GPU implementation are compared with timings of single-core CPU implementation. Operating frequency is chosen as 1 GHz, and the electrical size of the geometry is almost 52λ. The number of unknowns (RWG functions) used to discretize the geometry is 718347. Geometry is illuminated by two different linearly polarized plane-waves as horizontally polarized (y-polarized ([0,1,0]), h-pol in short) and vertically polarized (z-polarized ([0,0,1]), v-pol in short). Plane-waves are unit magnitude and propagating along the $+x$-direction ([1,0,0]) as shown in Figure 4.22. The resulting scattered fields are calculated along the $\theta = 90^\circ$ cut (xy-plane), where ϕ values vary from 0 to 360 degrees with 1 degree spacing (see the red circle shown in Figure 4.22). In the numerical integrations over the triangles, 3-point testing function and 4-point basis function are used for the near-field interactions, and 3-point testing function and 3-point basis function are used for the far-field interactions. For the CFIE formulation, $\alpha = 0.5$ is used . The problem is solved with 8 MLFMA level, and the total number of near-field interactions is 520071909. The number of total iterations is 330.

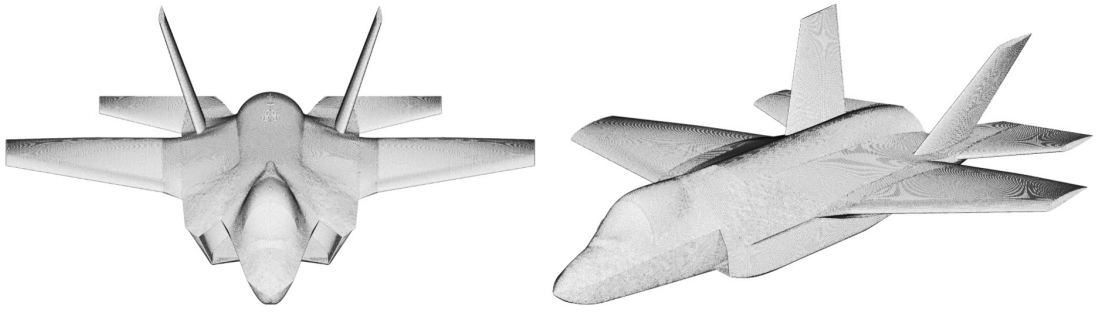
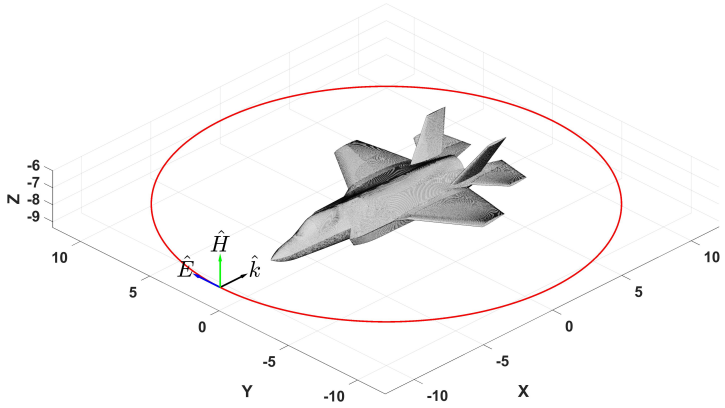
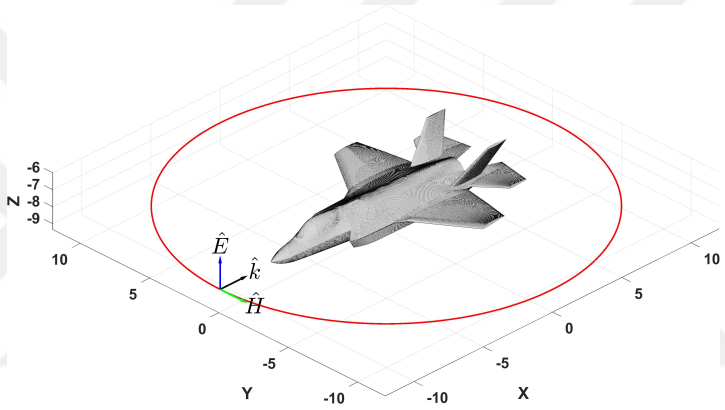


Figure 4.21: Aircraft Geometry

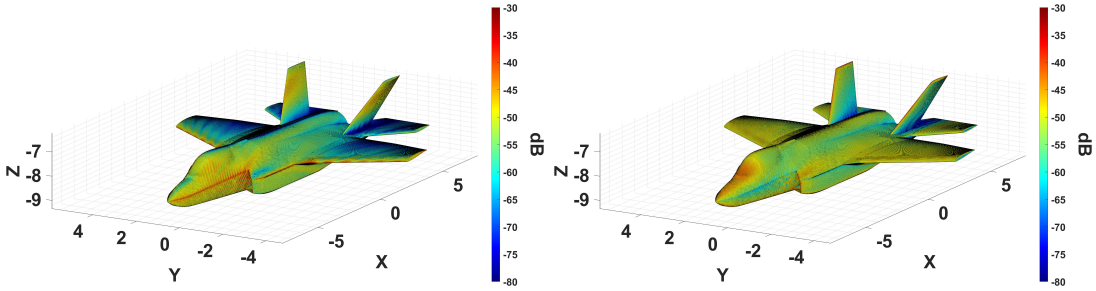


(a) H-pol



(b) V-pol

Figure 4.22: Illumination of the aircraft model with horizontal and vertical polarizations



(a) Illuminated by H-pol

(b) Illuminated by V-pol

Figure 4.23: Induced currents on the aircraft geometry

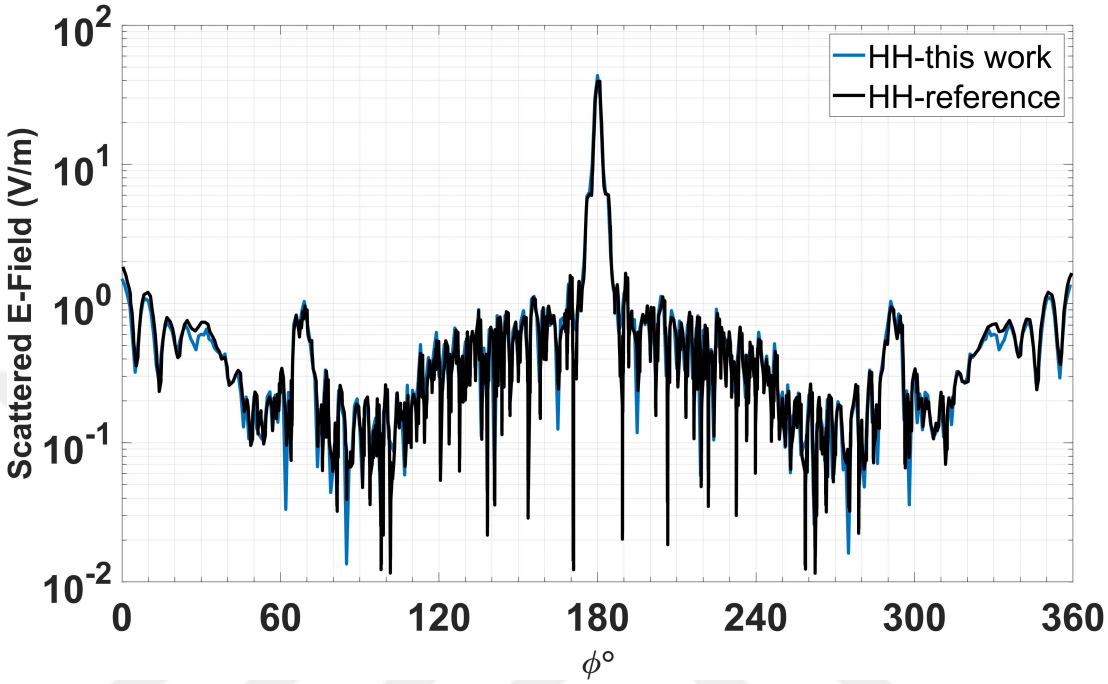


Figure 4.24: The comparison of scattered fields from the aircraft geometry by the reference [16]

The induced currents, for each illumination, are shown in Figure 4.23. The resulting scattered field for the HH-polarization (i.e., E_ϕ field under the horizontal plane-wave illumination) is compared with a reference solution available in the literature [16] to show the good agreement in Figure 4.24. Furthermore, in Figure 4.25 and Figure 4.26, all of the scattered fields can be observed for all polarizations (i.e., HH, HV, VV and VH) in the form of a polar graph.

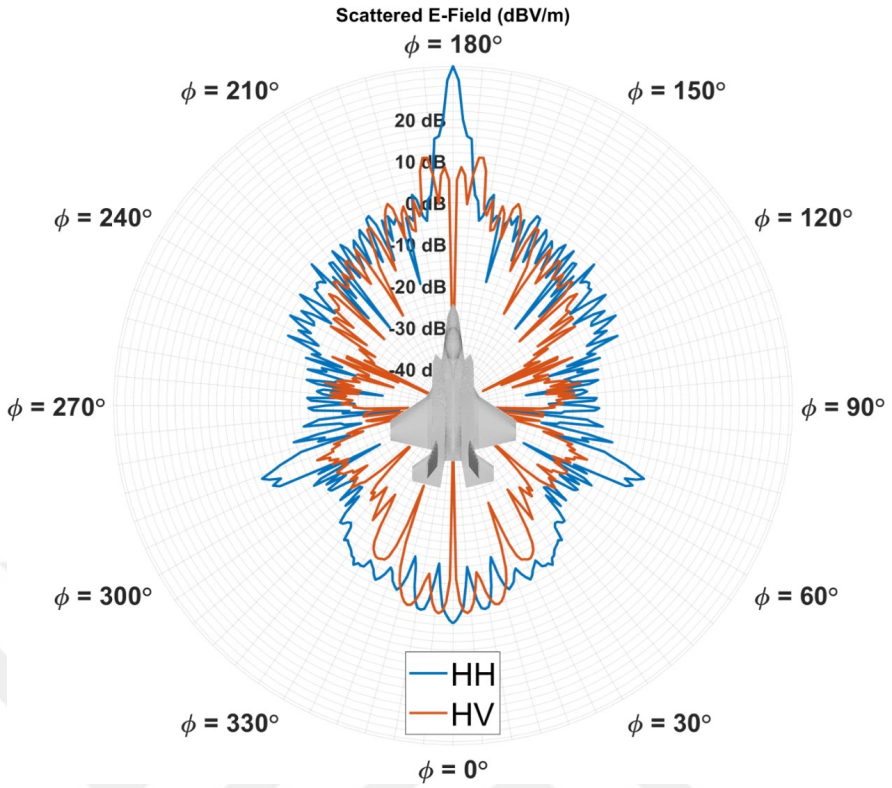


Figure 4.25: Scattered E-field of the aircraft geometry when illuminated by an H-pol plane wave

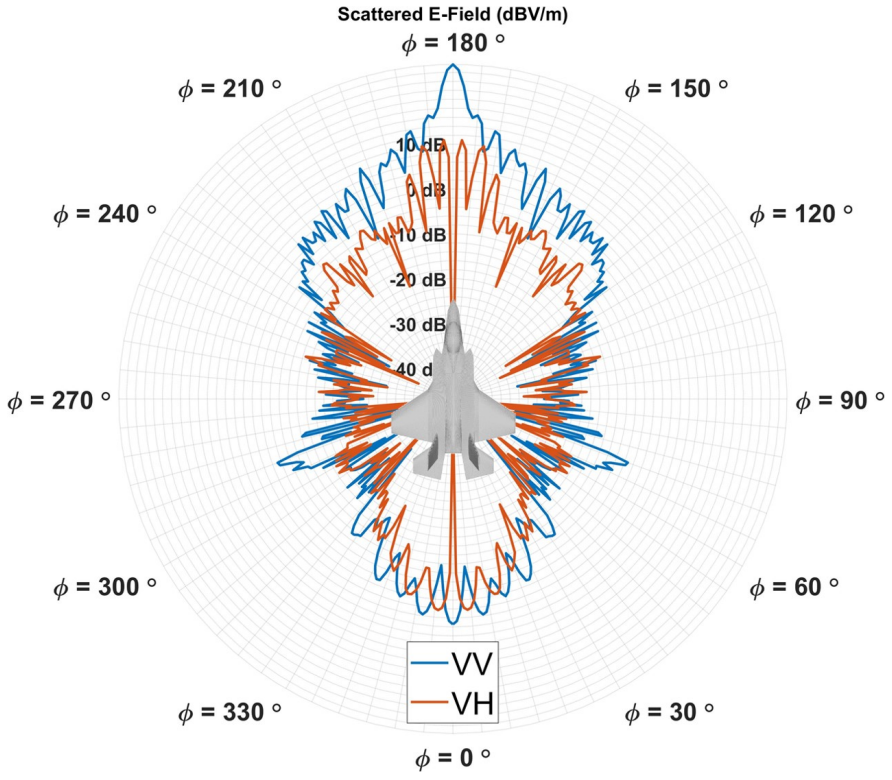


Figure 4.26: Scattered E-field of the aircraft geometry when illuminated by a V-pol plane wave

	Z_{near}	Lowest Level Agg.	Sub-Level Agg.	Translation	Sub-Level DisAgg	Lowest Level DisAgg.	Total MVM	Total
CPU	35636.69	77821.07	1944.55	7956.93	2005.73	82792.29	172520.57	208157.26
GPU	1975.49	14002.85	509.08	1722.27	538.85	15737.61	32510.66	34486.15
Speed-up	18.04	5.56	3.82	4.62	3.72	5.26	5.31	6.04

Table 4.6: Time comparison of CPU and GPU implementations for the aircraft geometry illuminated by the H-pol plane-wave

	Z_{near}	Lowest Level Agg.	Sub-Level Agg.	Translation	Sub-Level DisAgg	Lowest Level DisAgg.	Total MVM	Total
CPU	35502.66	71114.36	1670.96	7015.18	1722.2	70205.55	151728.25	187230.91
GPU	1976.06	11744.01	441.99	1453.19	462.33	13167.5	27269.02	29245.08
Speed-up	17.97	6.06	3.78	4.83	3.73	5.33	5.56	6.40

Table 4.7: Time comparison of CPU and GPU implementations for the aircraft geometry illuminated by the V-pol plane-wave

In Table 4.6 and Table 4.7, detailed timings of the aircraft analyses are given for H-pol and V-pol illuminations, respectively. For each part of MLFMA, timings are presented separately. Similar to the sphere analyses, the best efficiencies are obtained for the near-field part and the lowest-level aggregation/disaggregation stages with speed-up values of $\sim$18 to $\sim$5.5, respectively. The total time speed-up is 6 times and 6.4 times for H-pol and V-pol analyses, respectively. An important remark can be made about the speed-up values of translation stages because they are higher than the sub-level aggregation/disaggregation parts, whereas it is observed as the lowest enhancement in sphere analyses. This can be from the fact that the geometrical shape of an aircraft is very different from that of a sphere. This shape effect results in different far-field interaction schemes (i.e., number of boxes that remain in the far-field of other boxes according to the one-box buffer scheme), and this scheme can change the domination of distribution and multiplication processes which can directly affect the performance.

4.2.5 Missile Results

In this part, the results of MLFMA GPU implementation for an arbitrary missile geometry (as shown in Figure 4.27) are presented. The results are obtained on the server system. The timings of GPU implementation are compared with timings of single-core CPU implementation. Operating frequency is chosen as 1 GHz, and the electrical size of the geometry is almost 13λ. The number of unknowns (RWG functions) used to discretize the geometry is 31305. Geometry is illuminated by two different linearly polarized plane-waves as horizontally polarized (y-polarized ([0,1,0])) and vertically polarized (z-polarized ([0,0,1])). Plane-waves are unit magnitude and propagating along the $+x$-direction ([1,0,0]) as shown in Figure 4.28. The resulting scattered fields are calculated along the $\theta = 90^{\circ}$ cut (xy-plane), where ϕ values vary from 0 to 360 degrees with 1 degree spacing (see the red circle shown in Figure 4.28). In the numerical integrations over the triangles, 3-point testing function and 4-point basis function are used for the near-field interactions, and 3-point testing function and 3-point basis function are used for the far-field interactions. For the CFIE formulation, $\alpha = 0.5$ is used . The problem is solved in 6 level, and the total number of near-field interactions is 22238087. The number of total iterations is 117.

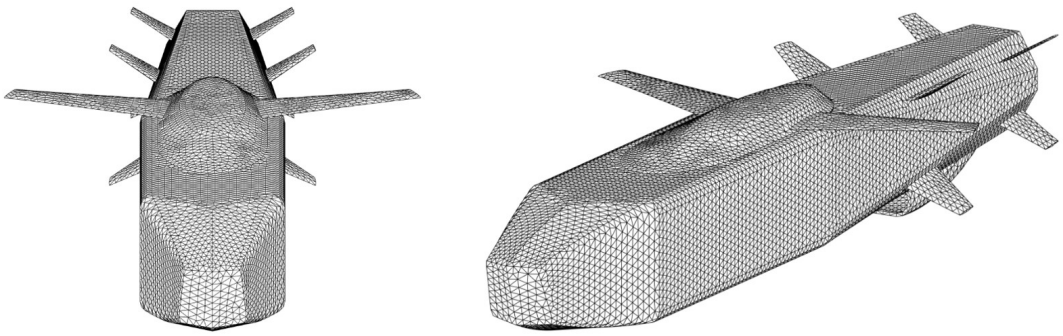
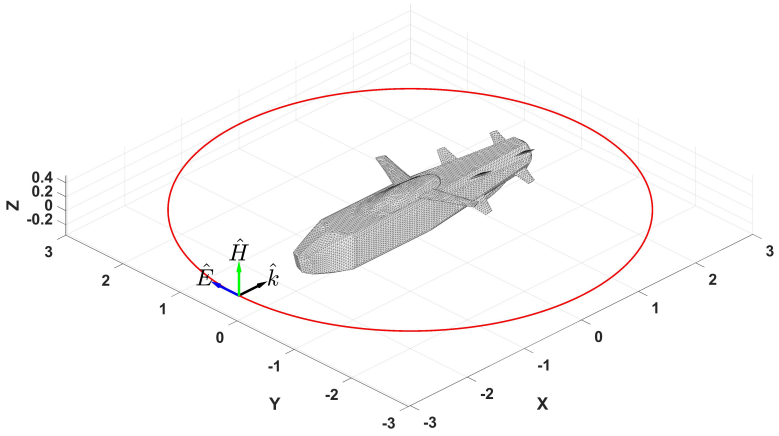
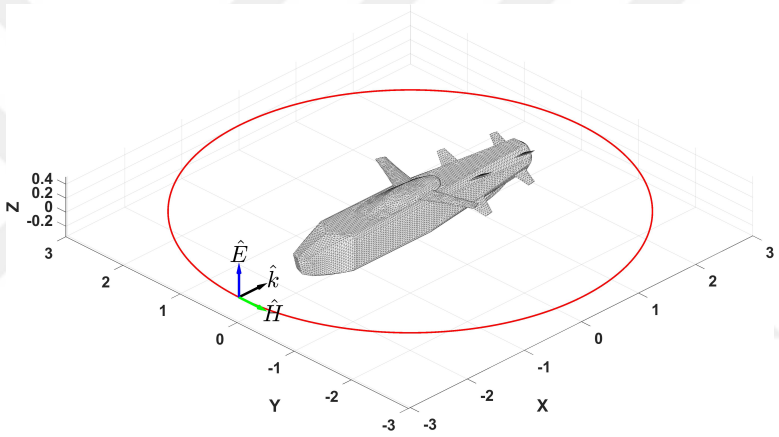


Figure 4.27: Missile geometry

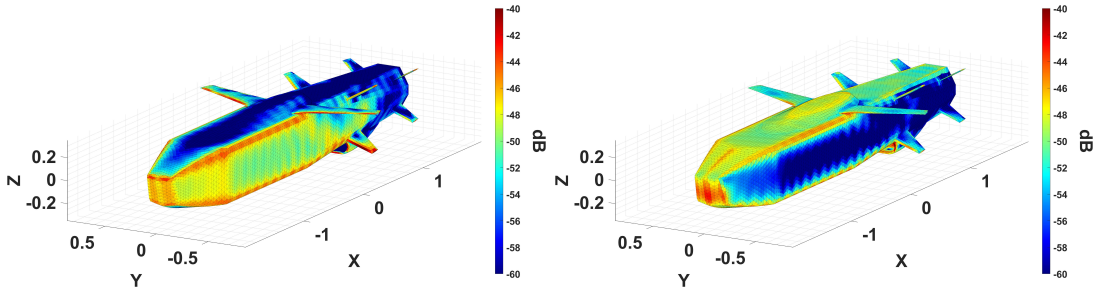


(a) H-pol



(b) V-pol

Figure 4.28: Illumination of the missile model with horizontal and vertical polarizations



(a) Induced currents on missile geometry H-pol

(b) Induced currents on missile geometry V-pol

Figure 4.29: Induced currents on missile geometry

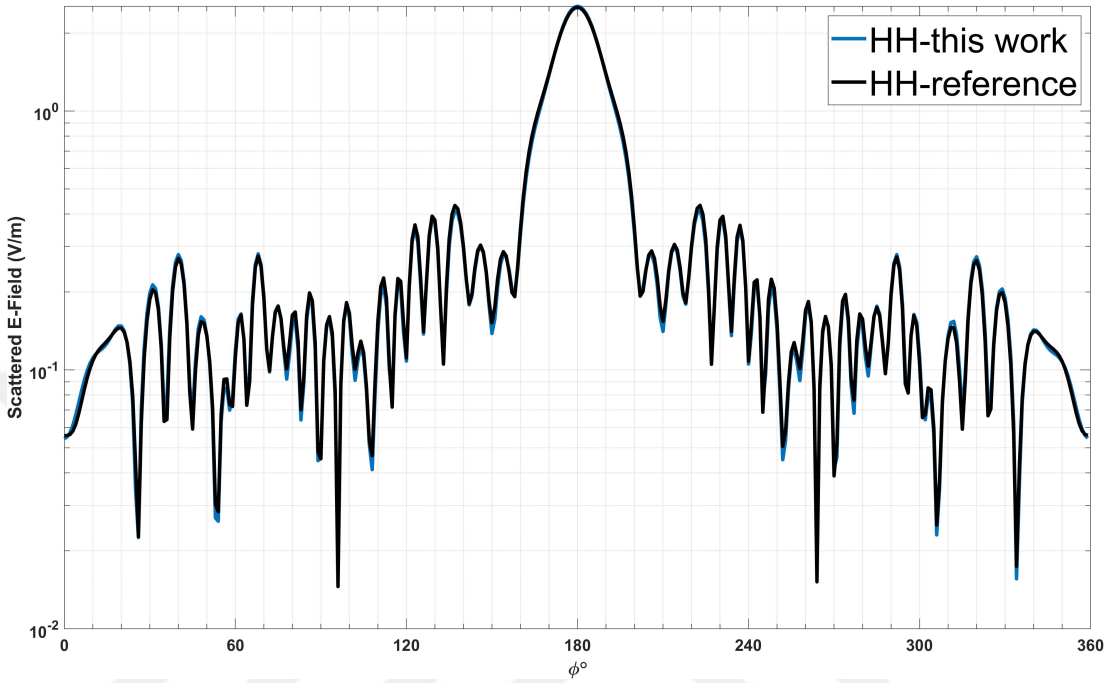


Figure 4.30: The comparison of scattered fields from the missile geometry with an independent reference

The induced currents, for each illumination, are shown in Figure 4.29. The resulting scattered field for the HH-polarization (i.e., E_ϕ field under the horizontal plane-wave illumination) is compared with an independent reference value to show the perfect agreement in Figure 4.30. Furthermore, in Figure 4.31 and Figure 4.32, all of the scattered fields can be observed for all polarization (i.e., HH, HV, VV and VH) in the form of a polar graph.

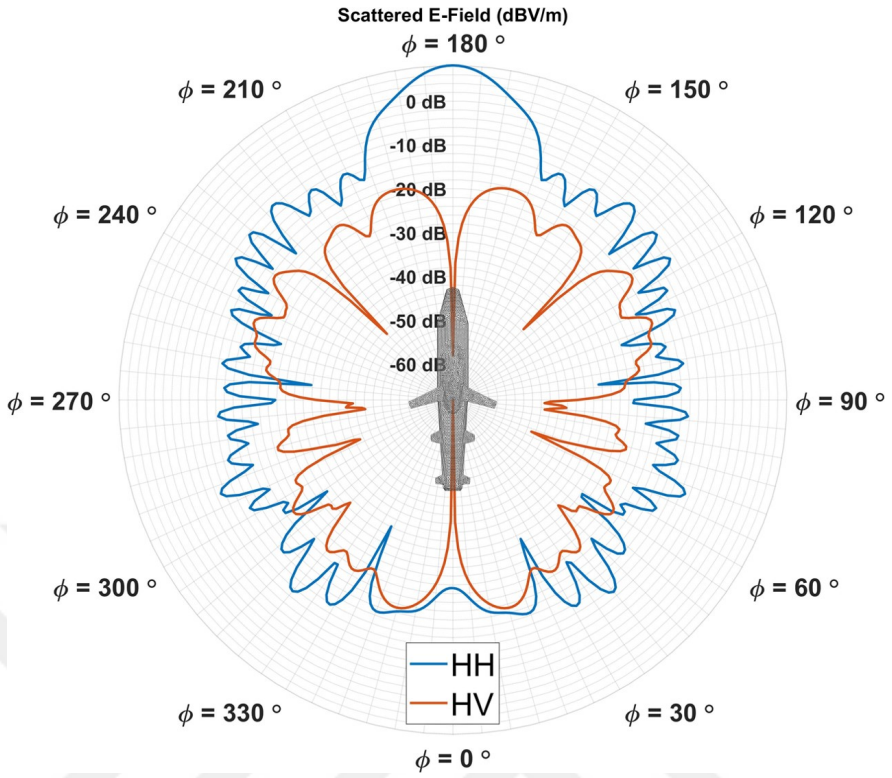


Figure 4.31: Scattered E-field from missile model polar plot

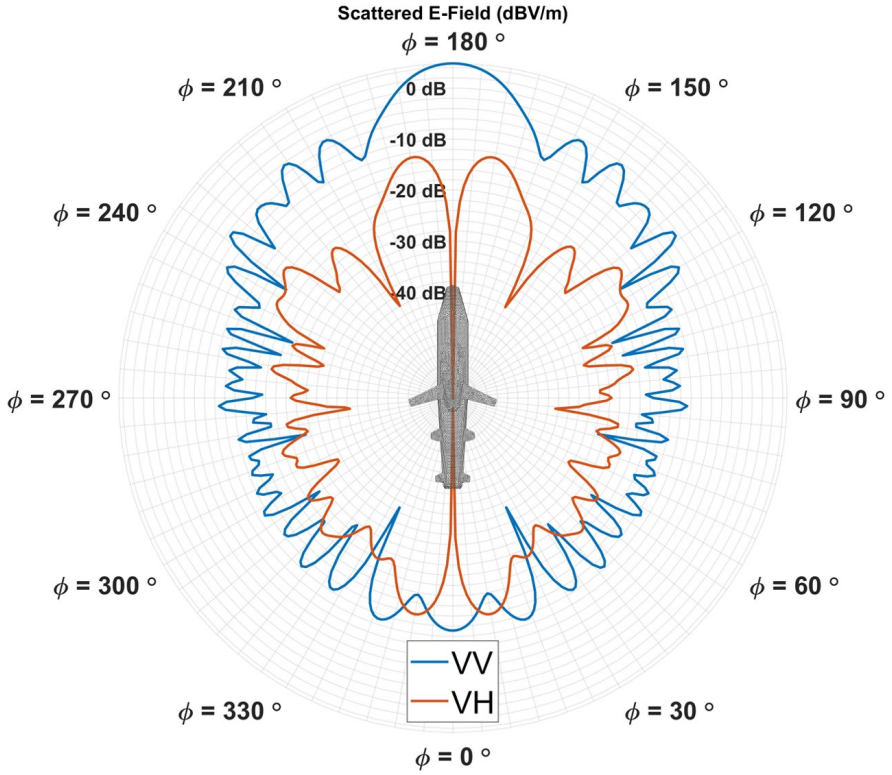


Figure 4.32: Scattered E-field from missile model polar plot

	Z_{near}	Lowest Level Agg.	Sub-Level Agg.	Translation	Sub-Level DisAgg	Lowest Level DisAgg.	Total MVM	Total
CPU	1277.97	985.79	14.5	54.69	13.27	1051.78	2120.05	3398.03
GPU	84.98	171.41	8.22	23.2	5.52	190.39	398.77	483.75
Speed-up	15.0380	5.7510	1.7630	2.3570	2.4022	5.5242	5.3165	7.0243

Table 4.8: Time comparison CPU vs. GPU for missile geometry illuminated by H-pol

	Z_{near}	Lowest Level Agg.	Sub-Level Agg.	Translation	Sub-Level DisAgg	Lowest Level DisAgg.	Total MVM	Total
CPU	1297.52	1004.35	13.53	54.25	12.82	1079.04	2164.01	3461.54
GPU	84.21	175.37	8.47	24.09	5.36	194.62	407.94	492.15
Speed-up	15.4	5.72	1.59	2.25	2.38	5.54	5.30	7.03

Table 4.9: Time comparison CPU vs. GPU for missile geometry illuminated by V-pol

In Table 4.8 and Table 4.9, detailed timings of the missile analyses are given for H-pol and V-pol illuminations, respectively. Similar to the aircraft and sphere analyses, the best efficiencies are obtained for the near-field part and the lowest-level aggregation/disaggregation stages with speed-up values of $\sim$15 to $\sim$5.5, respectively. The total speed-up is around 7 times for H-pol and V-pol analyses. The performance enhancement of translation stage can be explained by the shape effect of the geometry which directly affects the far-field interactions schemes and thus the SIMD compatibility of the translation operations as discussed in the previous Section 4.2.4 at page 87.

Chapter 5

Conclusion and Recommendations

In this thesis, a matrix partitioning algorithm is presented to solve large-scale EM scattering problems with the full-wave solver MLFMA on single-GPU systems. The matrix partitioning scheme is proposed to eliminate the RAM restriction of GPU devices to be able to solve large-scale EM problems while acquiring reasonable accelerations. In order to obtain an efficient implementation, the internal structure of GPU is examined, such as its multi-core scheme, memory capacities/bandwidths, and massively parallel processing power. Although GPUs have very high-performance processing power, GPU implementations are not straightforward, and the targeted mathematical processes must be compatible with the SIMD structure. Otherwise, it may result in even performance degradation. In other words, the targeted process must have two specifications: Firstly, it needs to be massively parallel that can be broken into many independent units of the same work. Secondly, it must have dense operational weight or be computationally intensive. Besides, the bottlenecks of GPU hardware must be taken into account, such as the limited RAM capacity, data transmission delays, and device/memory initialization delays.

In this context, different operators of MLFMA are examined separately, namely,

the near-field calculation, the lowest-level aggregation/disaggregation, the sub-level aggregation/disaggregation, and the translation stages. Each stage has a different internal mathematical structure and operational weight. Thus their expected efficiencies are also different. Depending on their dense operational weight and SIMD compatibility, the near-field calculation and the lowest-level aggregation/disaggregation stages are expected to be more compatible with GPU implementation. On the other hand, the translation stage is expected to have the lowest compatibility because of its high-memory access need. After that, to be able to solve large-scale problems with MLFMA, the matrix partitioning scheme is introduced as a possible remedy to overcome the RAM restriction of GPU. Matrix partitioning scheme consists of dividing the data structures of MLFMA into small pieces, and solving them one-by-one on the GPU side, in order. However, although the RAM restriction is eliminated by this remedy, additional data transfer costs and initialization delays appear as the entire process is completed with many repeated small processes. Therefore, different implementation approaches are offered for the matrix partitioning scheme when determining the package size of each partition to detect an optimum region. So that the different operators of MLFMA can be fine-tuned on the GPU scale.

Preliminary results showcase the potential accelerations of our proposed scheme while the RAM limitation of GPU is eliminated. The numerical results demonstrate the overall efficiency and validate the expected accelerations of our implementation. Large-scale EM problems can be solved up to the electrical size of 88λ with 8577765 unknowns. Moreover, remarkable accelerations are reported for the total implementation times as 5 to 10 times with respect to single-core CPU implementations on different hardware infrastructures, namely a PC and a server. The best-accelerated parts are the near-field assembling, and the lowest-level aggregation/disaggregation stages up to 16 and 5 times, respectively. In addition, the lowest acceleration performance is observed for the translation operator down to 1.03, and even some performance degradation is reported like 0.61 times.

In this study, we accelerate the full-wave solver MLFMA with GPU hardware for

EM scattering problems modeled by surface integral equations (SIEs), namely the electric field integral equation (EFIE), magnetic field integral equation (MFIE), and combined field integral equation (CFIE). Another important future research direction can be the GPU implementation of volume integral equations (VIEs). Accelerating VIE-based large-scale EM problems on GPU can be useful because the number of unknowns in VIE-based EM scattering problems is much higher than the same electrical size of SIE-based problems. Furthermore, different implementation strategies can be developed for the operators of MLFMA to improve the overall efficiency. Finally, machine-learning-based techniques can be integrated with GPU implementation to obtain more efficient solutions.

Bibliography

[1] C. A. Balanis, *Advanced engineering electromagnetics*. John Wiley & Sons, 2012.

[2] W. C. Gibson, *The method of moments in electromagnetics*. Chapman and Hall/CRC, 2021.

[3] O. Ergul and L. Gurel, *The multilevel fast multipole algorithm (MLFMA) for solving large-scale computational electromagnetics problems*. John Wiley & Sons, 2014.

[4] W. C. Chew, E. Michielssen, J. Song, and J.-M. Jin, *Fast and efficient algorithms in computational electromagnetics*. Artech House, Inc., 2001.

[5] B. Khalichi, *Fast and efficient solutions of multiscale electromagnetic problems*. PhD thesis, Bilkent University, 2020.

[6] R. F. Harrington, *Field computation by moment methods*. Wiley-IEEE Press, 1993.

[7] B. Khalichi, O. Ergül, and V. Ertürk, "Near-field-based preconditioning technique in the incomplete-leaf MLFMA for nonuniformly discretized electromagnetic scattering problems," in *2021 IEEE AP-S Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, 2021.

[8] B. Khalichi, O. Ergül, and V. Ertürk, "An integral-equation-based method for efficient and accurate solutions of scattering problems with highly nonuniform discretizations," in *2021 IEEE AP-S Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, 2021.

[9] B. Khalichi, Ö. Ergül, M. Takrimi, and V. B. Ertürk, "Broadband analysis of multiscale electromagnetic problems: Novel incomplete-leaf MLFMA for potential integral equations," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 12, pp. 9032–9037, 2021.

[10] B. Khalichi, V. B. Ertürk, and Ö. Ergül, "Fast solutions of multiscale electromagnetic problems using potential integral equations," in *2020 IEEE International Symposium on Antennas and Propagation and North American Radio Science Meeting*, pp. 1071–1072, IEEE, 2020.

[11] B. Khalichi, Ö. Ergül, and V. B. Ertürk, "Broadband solutions of potential integral equations with NSPWMLFMA," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 6, pp. 4307–4312, 2019.

[12] B. Khalichi, Ö. Ergül, M. Takrimi, and V. B. Ertürk, "Solution of potential integral equations with NSPWMLFMA," in *2018 IEEE International Symposium on Antennas and Propagation & USNC/URSI National Radio Science Meeting*, pp. 2307–2308, IEEE, 2018.

[13] B. Karaosmanoğlu and Ö. Ergül, "Acceleration of MLFMA simulations using trimmed tree structures," *IEEE Transactions on Antennas and Propagation*, vol. 69, no. 1, pp. 356–365, 2020.

[14] B. Karaosmanoğlu and Ö. Ergül, "Visual result prediction in electromagnetic simulations using machine learning," *IEEE Antennas and Wireless Propagation Letters*, vol. 18, no. 11, pp. 2264–2266, 2019.

[15] B. Karaosmanoğlu and Ö. Ergül, "Error prediction in electromagnetic simulations using machine learning," in *2019 IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, pp. 787–788, IEEE, 2019.

[16] B. Karaosmanoğlu, "Electromagnetic interaction complexity reduction using deep learning," 2019.

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

[18] N. Engheta, W. D. Murphy, V. Rokhlin, and M. S. Vassiliou, "The fast multipole method (FMM) for electromagnetic scattering problems," *IEEE Transactions on Antennas and Propagation*, vol. 40, no. 6, pp. 634–641, 1992.

[19] B. Khalichi, O. Ergül, and V. Ertürk, "Error analysis of QR-decomposition-based nondirective stable plane wave MLFMA," in *2018 IEEE AP-S Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting*, 2018.

[20] E. Wu and Y. Liu, "Emerging technology about GPGPU," in *APCCAS 2008-2008 IEEE Asia Pacific Conference on Circuits and Systems*, pp. 618–622, IEEE, 2008.

[21] A. Benhamida, M. Kozlovszky, and S. Szénási, "GPU usage trends in medical image processing," in *2019 IEEE 13th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pp. 320–325, IEEE, 2019.

[22] B. Neelima and P. S. Raghavendra, "Recent trends in software and hardware for GPGPU computing: a comprehensive survey," in *2010 5th International Conference on Industrial and Information Systems*, pp. 319–324, IEEE, 2010.

[23] C. A. Navarro, N. Hitschfeld-Kahler, and L. Mateu, "A survey on parallel computing and its applications in data-parallel problems using GPU architectures," *Communications in Computational Physics*, vol. 15, no. 2, pp. 285–329, 2014.

[24] E. Lezar and D. Davidson, "GPU acceleration of method of moments matrix assembly using Rao-Wilton-Glisson basis functions," in *2010 International Conference on Electronics and Information Engineering*, vol. 1, pp. V1–56, IEEE, 2010.

[25] N. A. Gumerov and R. Duraiswami, "Fast multipole methods on graphics processors," *Journal of Computational Physics*, vol. 227, no. 18, pp. 8290–8313, 2008.

[26] C. Wagenaar, "Acceleration of the multi level fast multipole algorithm on a graphics processing unit," 2016.

[27] 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," *IEEE transactions on antennas and propagation*, vol. 61, no. 7, pp. 3607–3616, 2013.

[28] M. Cwikla, J. Aronsson, and V. Okhmatovski, "Low-frequency MLFMA on graphics processors," *IEEE Antennas and Wireless Propagation Letters*, vol. 9, pp. 8–11, 2010.

[29] E. García, C. Delgado, L. Lozano, and F. Cátedra, "Efficient strategy for parallelisation of multilevel fast multipole algorithm using CUDA," *IET Microwaves, Antennas & Propagation*, vol. 13, no. 10, pp. 1554–1563, 2019.

[30] K. Xu, D. Z. Ding, Z. H. Fan, and R. S. Chen, "Multilevel fast multipole algorithm enhanced by GPU parallel technique for electromagnetic scattering problems," *Microwave and Optical Technology Letters*, vol. 52, no. 3, pp. 502–507, 2010.

[31] J. Guan, S. Yan, and J.-M. Jin, "An FE-BI-MLFMA with GPU acceleration for electromagnetic scattering analysis," in *2014 IEEE Antennas and Propagation Society International Symposium (APSURSI)*, pp. 747–748, IEEE, 2014.

[32] J. Guan, S. Yan, and J.-M. Jin, "An accurate and efficient finite element-boundary integral method with GPU acceleration for 3-D electromagnetic analysis," *IEEE transactions on antennas and propagation*, vol. 62, no. 12, pp. 6325–6336, 2014.

[33] Z. He and R. S. Chen, "GPU-accelerated ADI-PE method for analysis of EM scatterings," *Electronics Letters*, vol. 51, no. 21, pp. 1652–1654, 2015.

[34] N. Tran and O. Kilic, "Parallel implementations of multilevel fast multipole algorithm on graphical processing unit cluster for large-scale electromagnetics objects.," *Applied Computational Electromagnetics Society Journal*, vol. 33, no. 2, 2018.

[35] T. Phan, N. Tran, and O. Kilic, "Multi-level fast multipole algorithm for 3-D homogeneous dielectric objects using MPI-CUDA on GPU cluster.," *Applied Computational Electromagnetics Society Journal*, vol. 33, no. 3, 2018.

[36] S. Peng and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 7, pp. 2130–2133, 2008.

[37] S. Peng and Z. Nie, "Acceleration of the method of moments calculations by using graphics processing units," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 7, pp. 2130–2133, 2008.

[38] Z. Dan, Z. Rongcai, H. Lin, W. Tao, *et al.*, "A parallelization cost model for GPU," in *2010 International Conference on Computer and Communication Technologies in Agriculture Engineering*, vol. 3, pp. 515–519, IEEE, 2010.

[39] J. Owens, "GPU architecture overview," in *ACM SIGGRAPH 2007 courses*, pp. 2–es, 2007.

[40] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, vol. 7, no. 3, pp. 856–869, 1986.

[41] S. Schelkunoff, "Some equivalence theorems of electromagnetics and their application to radiation problems," *The Bell System Technical Journal*, vol. 15, no. 1, pp. 92–112, 1936.

[42] L. Correia, "A comparison of integral equations with unique solution in the resonance region for scattering by conducting bodies," *IEEE transactions on antennas and propagation*, vol. 41, no. 1, pp. 52–58, 1993.

[43] S. Rao, D. Wilton, and A. Glisson, "Electromagnetic scattering by surfaces of arbitrary shape," *IEEE Transactions on antennas and propagation*, vol. 30, no. 3, pp. 409–418, 1982.

[44] G. Cowper, "Gaussian quadrature formulas for triangles," *International Journal for Numerical Methods in Engineering*, vol. 7, no. 3, pp. 405–408, 1973.

[45] D. Wilton, S. Rao, A. Glisson, D. Schaubert, O. Al-Bundak, and C. Butler, "Potential integrals for uniform and linear source distributions on polygonal and polyhedral domains," *IEEE Transactions on Antennas and Propagation*, vol. 32, no. 3, pp. 276–281, 1984.

[46] R. D. Graglia, "On the numerical integration of the linear shape functions times the 3-d green's function or its gradient on a plane triangle," *IEEE transactions on antennas and propagation*, vol. 41, no. 10, pp. 1448–1455, 1993.

[47] R. E. Hodges and Y. Rahmat-Samii, "The evaluation of MFIE integrals with the use of vector triangle basis functions," *Microwave and Optical Technology Letters*, vol. 14, no. 1, pp. 9–14, 1997.

[48] S. Koc, J. Song, and W. C. Chew, "Error analysis for the numerical evaluation of the diagonal forms of the scalar spherical addition theorem," *SIAM journal on numerical analysis*, vol. 36, no. 3, pp. 906–921, 1999.

[49] J. Song, C.-C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects," *IEEE Transactions on Antennas and Propagation*, vol. 45, no. 10, pp. 1488–1493, 1997.

[50] N. Zhang, Y.-s. Chen, and J.-l. Wang, "Image parallel processing based on GPU," in *2010 2nd international conference on advanced computer control*, vol. 3, pp. 367–370, IEEE, 2010.

[51] J. Breitbart, "Case studies on GPU usage and data structure design," *Master's thesis, Universität Kassel*, 2008.

[52] D. Lustig and M. Martonosi, "Reducing GPU offload latency via fine-grained CPU-GPU synchronization," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 354–365, IEEE, 2013.

[53] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "GPGPU processing in CUDA architecture," *arXiv preprint arXiv:1202.4347*, 2012.

[54] C. McClanahan, "History and evolution of GPU architecture," *A Survey Paper*, vol. 9, 2010.

[55] "Cuda c++ programming guide."

[56] "List of nvidia graphics processing units," Jan 2022.

[57] "GPU programming in matlab."

[58] S. Tang, B.-S. Lee, and B. He, "Speedup for multi-level parallel computing," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pp. 537–546, IEEE, 2012.

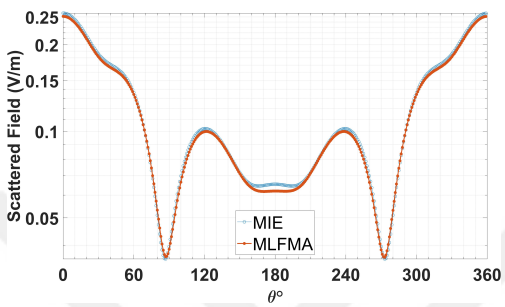
[59] M. Hidayetoglu, L. Gürel, *et al.*, "Reducing MLFMA memory with out-of-core implementation and data-structure parallelization," in *CEM'13 Computational Electromagnetics International Workshop*, pp. 34–37, IEEE, 2013.

[60] M. Hidayetoğlu and L. Gürel, "MLFMA memory reduction techniques for solving large-scale problems," in *2014 IEEE Antennas and Propagation Society International Symposium (APSURSI)*, pp. 749–750, IEEE, 2014.

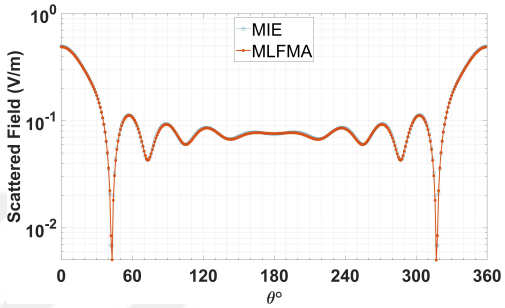
Appendix A

Scattered Fields of Sphere Analyses

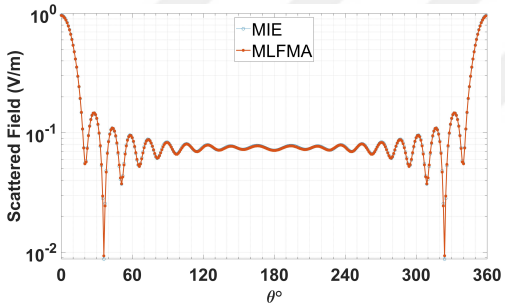
A.1 Results solved on PC



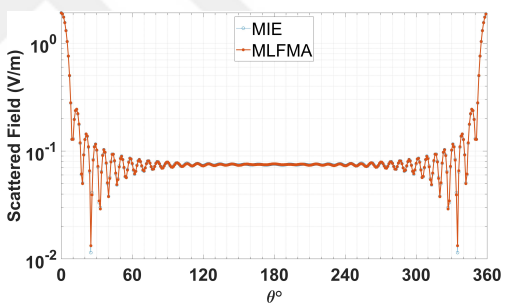
(a) 1λ sphere



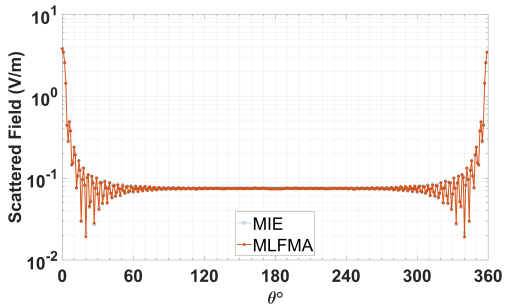
(b) 2λ sphere



(c) 4λ sphere

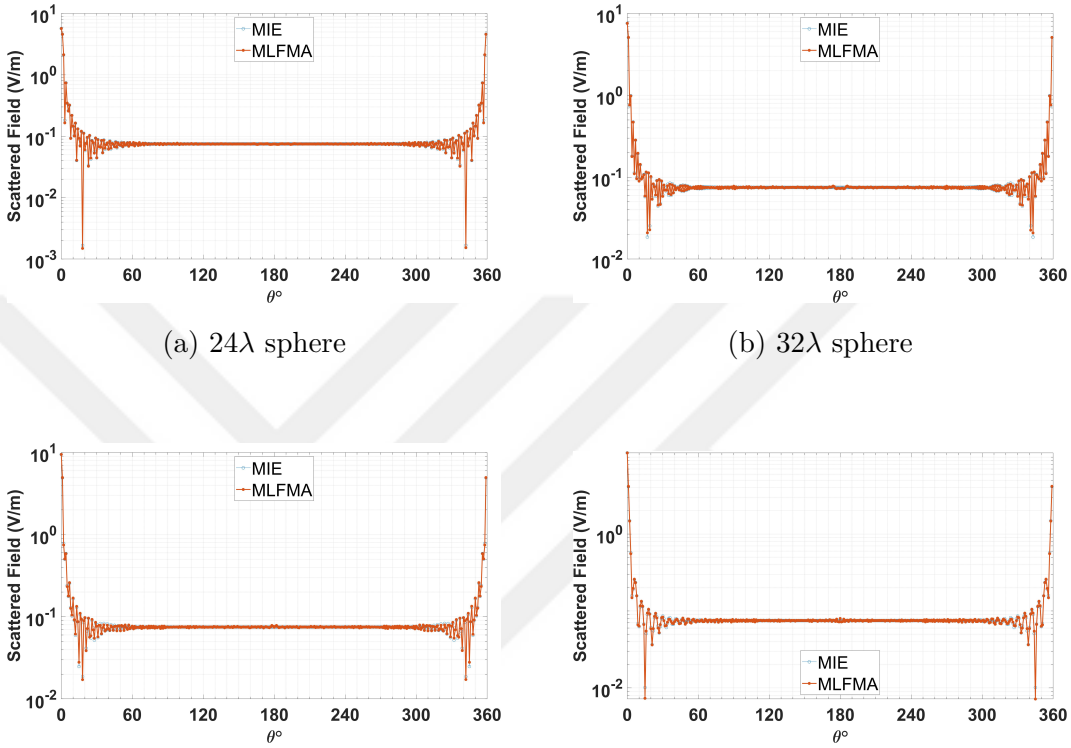


(d) 8λ sphere



(e) 16λ sphere

Figure A.1: The resulting scattered fields compared with MIE series solution for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on PC



(a) 24λ sphere

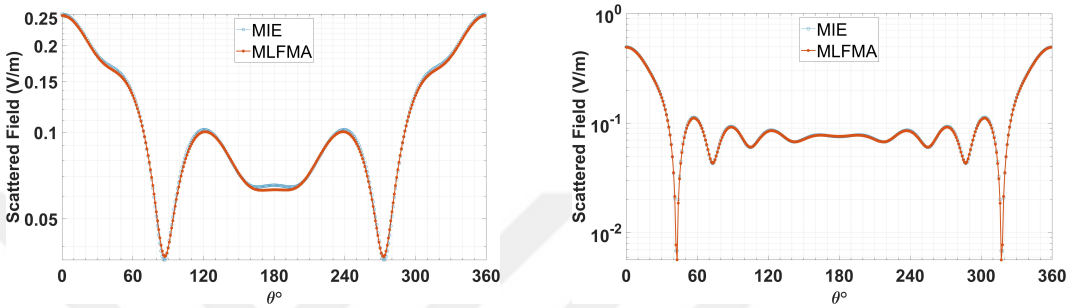
(b) 32λ sphere

(c) 40λ sphere

(d) 48λ sphere

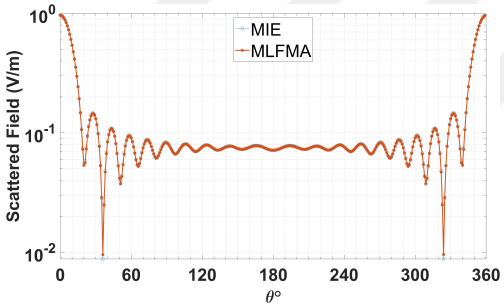
Figure A.2: The resulting scattered fields compared with MIE series solution for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda$ spheres solved on PC

A.2 Results solved on the server

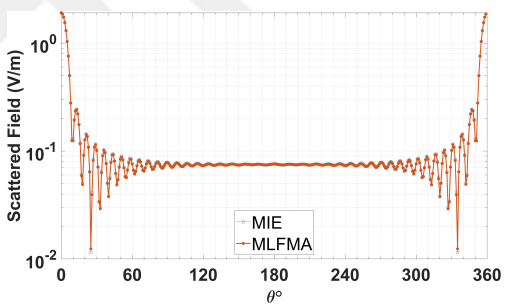


(a) 1λ sphere

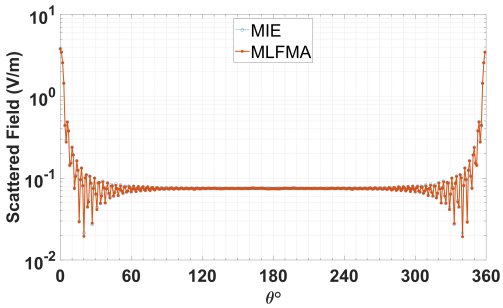
(b) 2λ sphere



(c) 4λ sphere

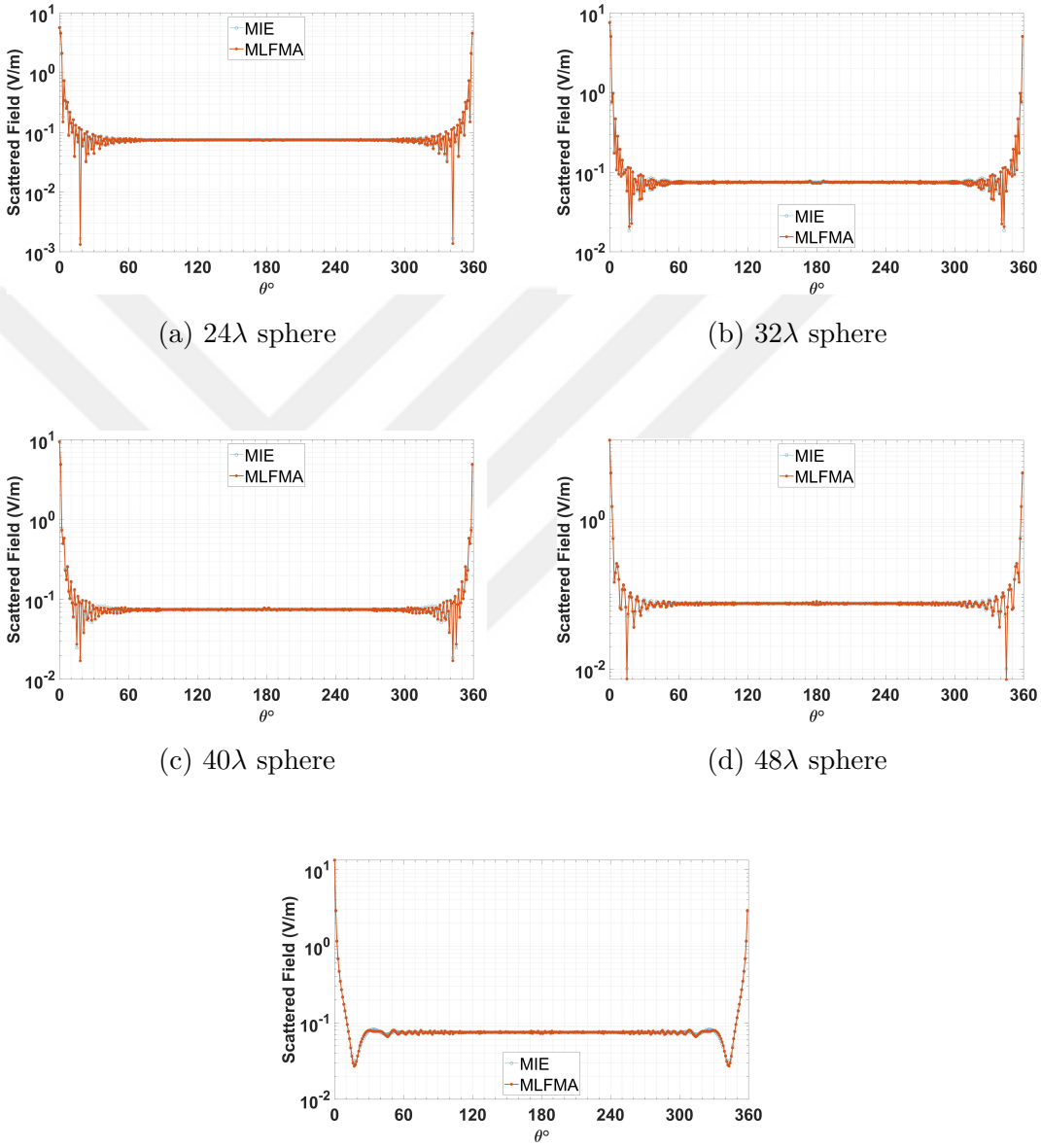


(d) 8λ sphere



(e) 16λ sphere

Figure A.3: The resulting scattered fields compared with MIE series solution for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on the server



(a) 24λ sphere

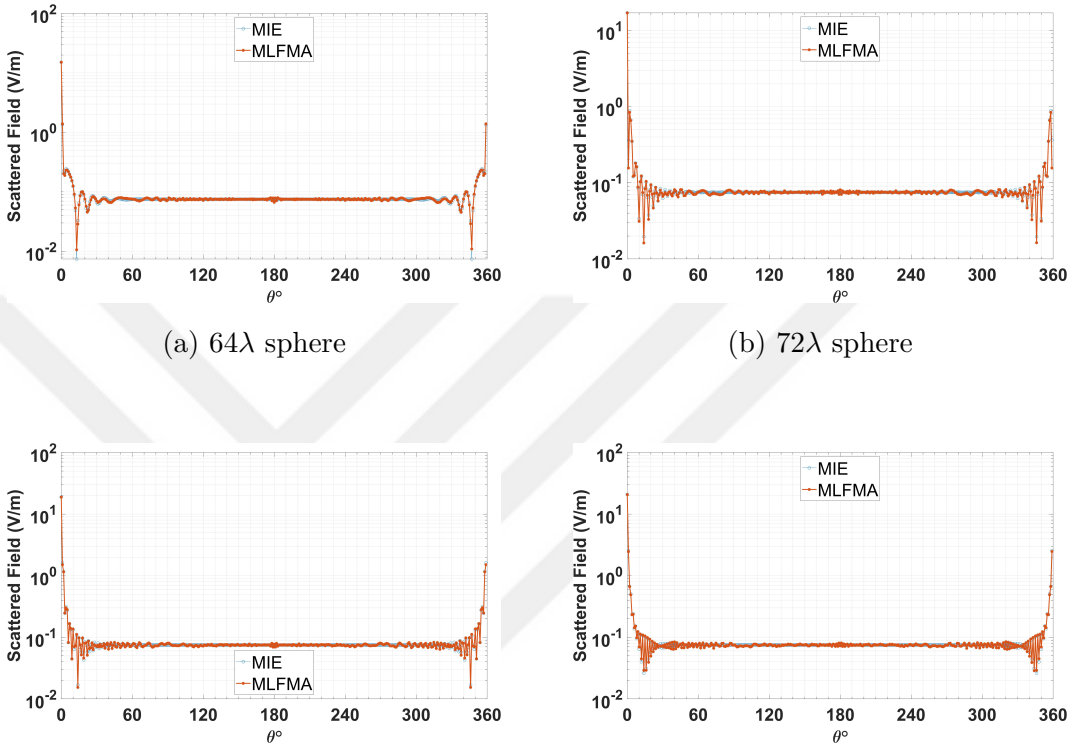
(b) 32λ sphere

(c) 40λ sphere

(d) 48λ sphere

(e) 56λ sphere

Figure A.4: The resulting scattered fields compared with MIE series solution for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda, 56\lambda$ spheres solved on the server



(a) 64λ sphere

(b) 72λ sphere

(c) 80λ sphere

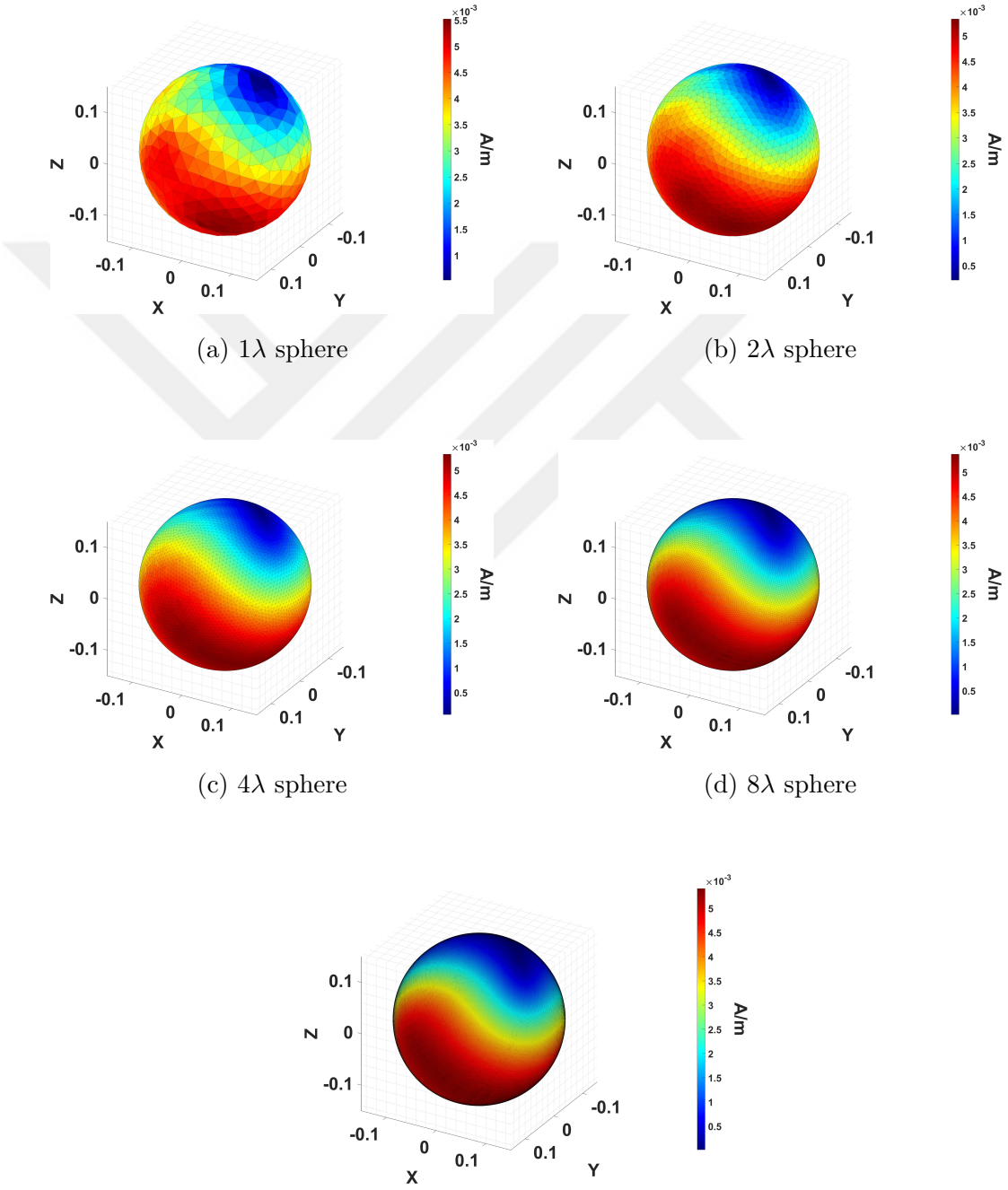
(d) 88λ sphere

Figure A.5: The resulting scattered fields compared with MIE series solution for the sphere analyses of $64\lambda, 72\lambda, 80\lambda, 88\lambda$ spheres solved on the server

Appendix B

Current Distributions of Sphere Analyses

B.1 Results solved on PC



(a) 1λ sphere

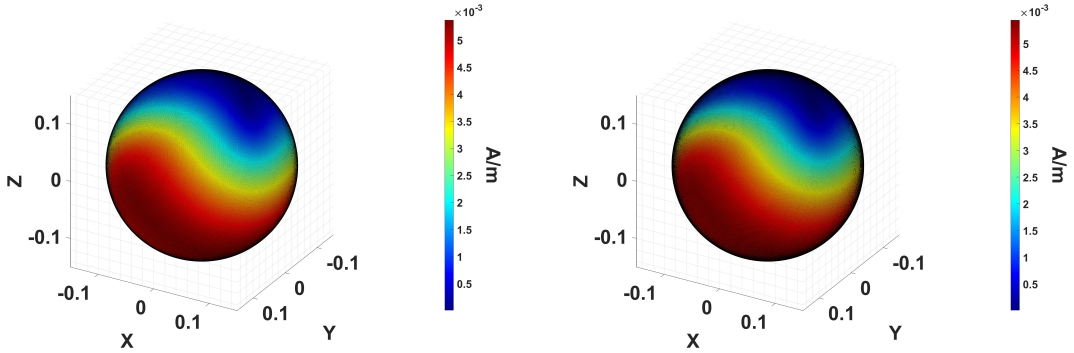
(b) 2λ sphere

(c) 4λ sphere

(d) 8λ sphere

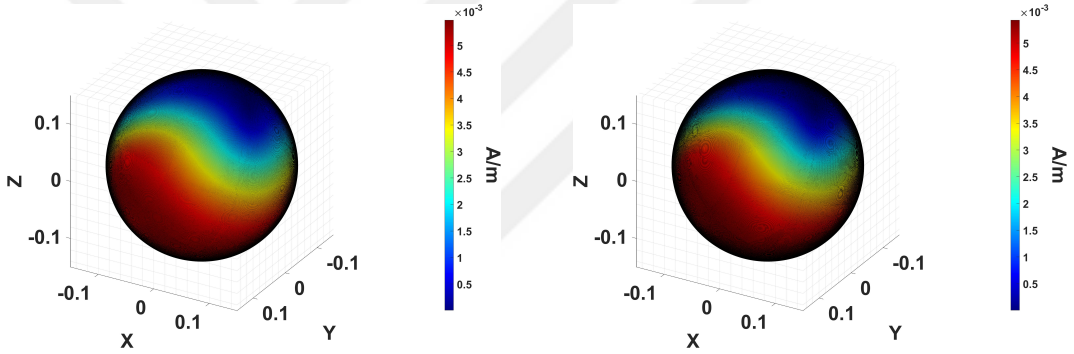
(e) 16λ sphere

Figure B.1: The resulting induced currents for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on PC



(a) 24λ sphere

(b) 32λ sphere

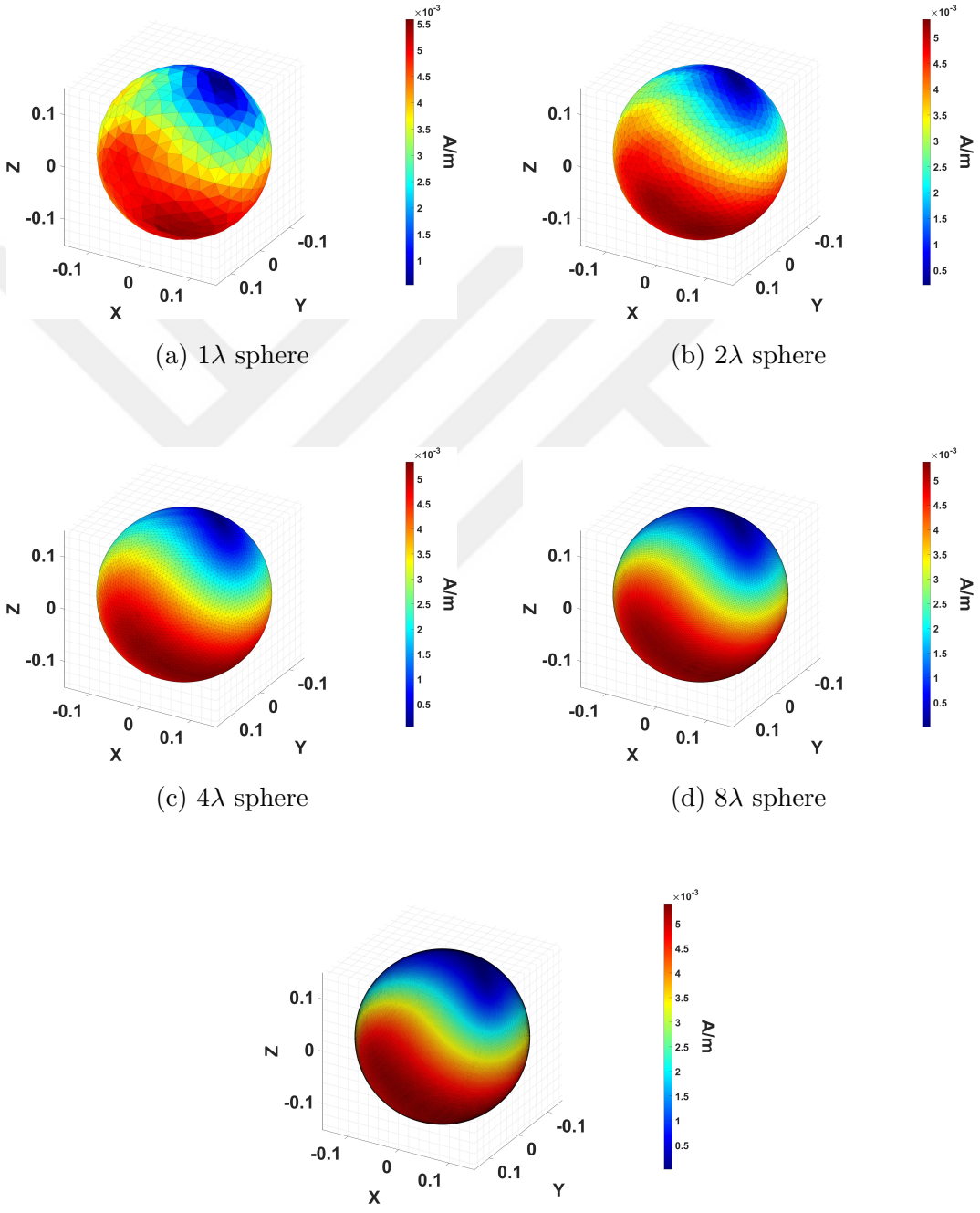


(c) 40λ sphere

(d) 48λ sphere

Figure B.2: The resulting induced currents for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda$ spheres solved on PC

B.2 Results solved on the server



(a) 1λ sphere

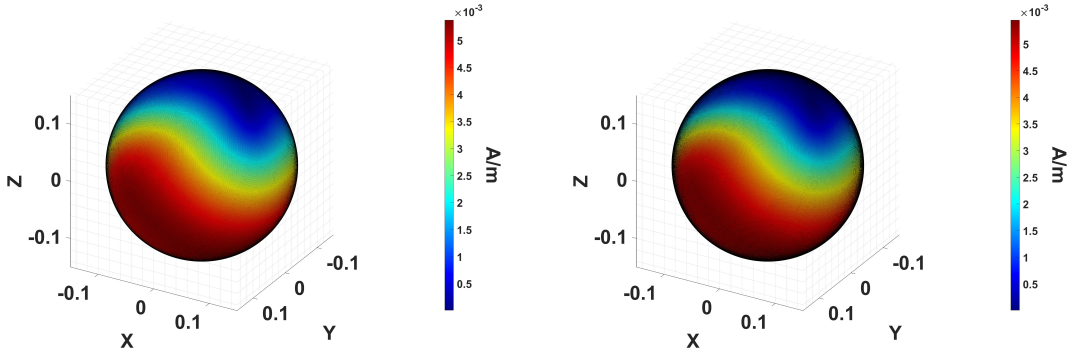
(b) 2λ sphere

(c) 4λ sphere

(d) 8λ sphere

(e) 16λ sphere

Figure B.3: The resulting induced currents for the sphere analyses of $1\lambda, 2\lambda, 4\lambda, 8\lambda, 16\lambda$ spheres solved on the server



(a) 1λ sphere

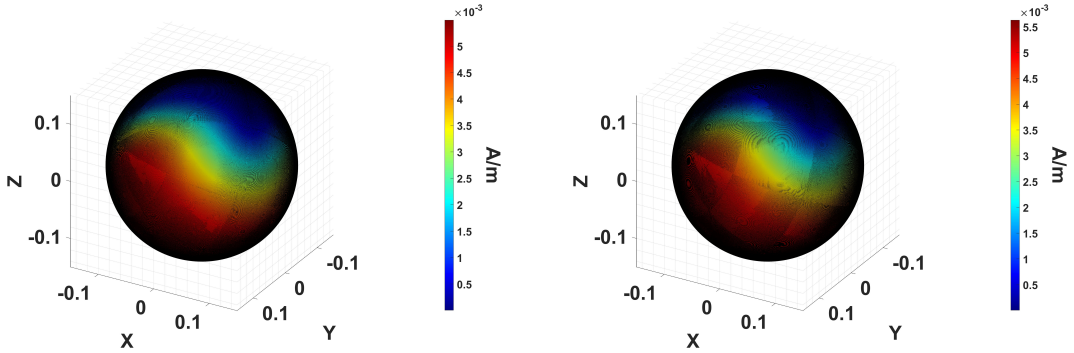
(b) 32λ sphere

(c) 40λ sphere

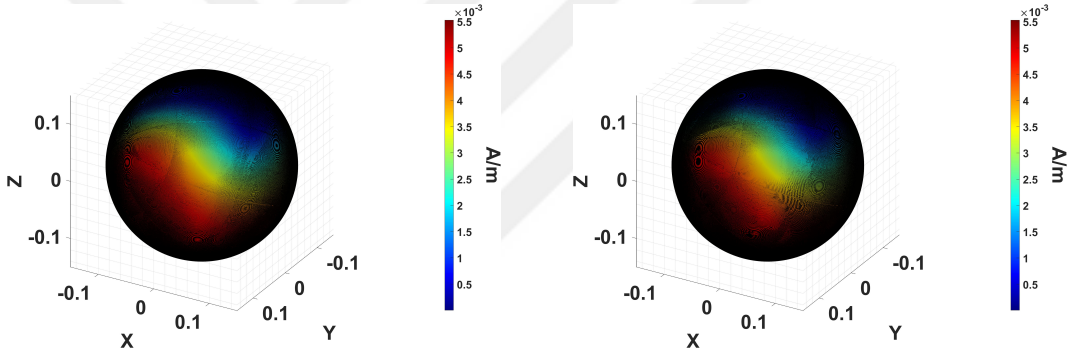
(d) 48λ sphere

(e) 56λ sphere

Figure B.4: The resulting induced currents for the sphere analyses of $24\lambda, 32\lambda, 40\lambda, 48\lambda, 56\lambda$ spheres solved on the server



(a) 64λ sphere (b) 72λ sphere



(c) 80λ sphere (d) 88λ sphere

Figure B.5: The resulting induced currents for the sphere analyses of $64\lambda, 72\lambda, 80\lambda, 88\lambda$ spheres solved on the server