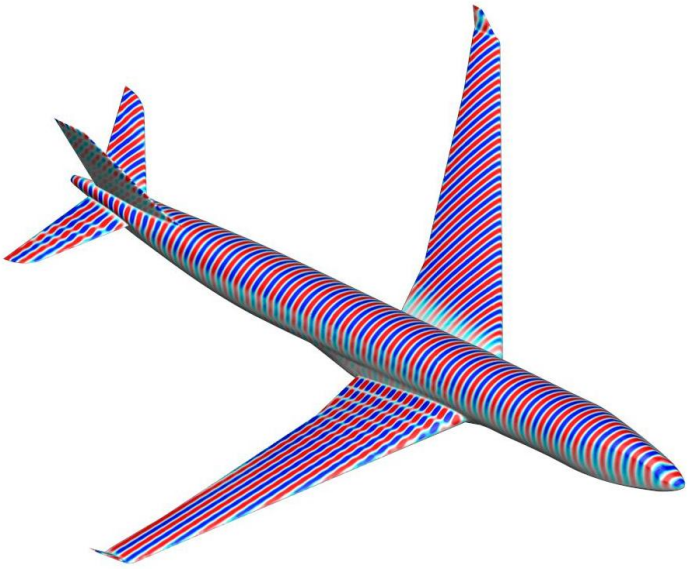
Thesis National Aerospace Laboratory NLR

Acceleration of the Multi Level Fast Multipole Method on a GPU



Author:
Claudia Wagenaar(4090772)

Supervisors:
H. van der Ven
D. van der Heul
C. Vuik

TU Delft

nlr

February 1, 2016

Abstract

In many circumstances, it is of great importance that enemy aircraft can be detected by a radar. Good knowledge about your own detectability is also needed to make your own visibility as low as possible. Obtaining the radar signature involves solving a scattering problem with a large, low-observable structure analysis. Briefly explained, the incident radar wave induces a current distribution on the surface of the scatterer. Using the Method of Moments (MoM), this current distribution can be found as the solution of an integral equation, which is one of the possibilities of obtaining this current distribution. Because every pair of basis functions interacts through the Green's function, after discretization it becomes a dense system matrix. For practical applications this system matrix is way too large to store. The Fast Multipole Method (FMM) became popular because it makes it possible to greatly reduce the memory storage and the work needed to solve the discretized integral equation. The scale of the work is decreased from $O(N^2)$ to $O(N^{\frac{3}{2}})$, where N is the number of unknowns. The Multi Level Fast Multipole Algorithm (MLFMA) reduces the required memory and computational complexity even more to $O(NlogN)$ by having different levels of clustering.

Radar signature computations are, with the use of the MLFMA, still computationally intensive. Therefore, Graphical Processor Units (GPU's) are sometimes used to try to improve the turn-around time by peforming calculations in parallel. Some parts of the MLFMA will be better suited for calculations on GPU's than others. Only by perfectly understanding the properties of GPU's, the applicability of the MLFMA can be fully exploited.

It turns out that some ways to accelerate the MLFMA are already discussed in the literature, which can be applied directly for the research part of this thesis. Besides, other possibilities will be explored for the specific MLFMA of the NLR. In literature, examples of the FMM and the MLFMA are discussed. In the algorithms in the literature, the focus is on the acceleration of the construction of the near field interaction matrix, the construction of the radiation and receiving patterns and the solve stage. The main focus for the research part of this thesis will be the acceleration of the solving part of the MLFMA.

Contents

Chapter 1

Introduction

Radar signature computations are very important but very time consuming to perform. Radars have been used to detect enemy aircraft since World War I. Ever since, knowing the radar signature of an aircraft, being friend or foe, has been used for modern warfare. The extent to which aircraft are detected by a radar is called the radar signature of an aircraft. There are two ways of obtaining this radar signature; experimentally or using very advanced computational methods. Those methods should become better and better at meeting the ever increasing demand for analyzing larger and more intricate platforms.
For simplicity, in this thesis it is assumped a single radar sends a plane wave of a single frequency to the aircraft. The idea of radar signature computations is to determine the equivalent current induced by the radar wave on the airplane. This current can then be used to determine the electromagnetic waves, reflected from the aircraft in any direction. The current distribution on the airplane induces an electric field somewhere else on the airplane. The current is approximated using a finite element method, where the unknowns are the scatterers or the degrees of freedom. The current distribution can be found as the solution of an integral equation. The integral equation is derived directly from the Maxwell equations, and therefore describes all the physics: it is a full-wave method. After discretization of the integral equation, the problem can be written as a matrix equation in the form $Ax = b$, where A is the interaction matrix between the basis functions, x are the currents on the airplane and b the incoming radar wave. Because of the size of many problems with a number of unknowns in the order of 10^7, this complete matrix can not be stored. The Multi Level Fast Multipole Algorithm (MLFMA) does not store the whole matrix and this is the reason why the algorithm became popular.

The MLFMA is still very time consuming and therefore other ways of accelerating the algorithm should be used. In this thesis Graphical Processor Units (GPU's) will be used to accelerate the algorithm MLFMA. GPU's perform calculations in parallel such that the result is obtained faster. The suitability of the building blocks of the MLFMA should be assessed one by one to see if they are suitable for the use of GPU's.

First, a research question will be composed, which can be answered at the end of the research part of the thesis. Afterwards the physical model will be told in more detail. In Chapter 4 the Fast Multipole Method (FMM) and the MLFMA will be explained in detail and the building blocks of the algorithm will be given. Then, the characteristics of GPU's are explained and possible bottlenecks will be discussed. In Chapter 6 some first ideas of the MLFMA on GPU's will be given and afterwards different ideas of the MLFMA on GPU's from the literature will be discussed. Finally, conclusions about the suitability of specific components for the algorithm

for acceleration with GPU's will be drawn and recommendations for the further research of this thesis will be given.

Chapter 2

Assessing the building blocks of the MLFMA for GPU acceleration

As already told in the introduction, it is very important that an enemy can be detected by a radar, but that your own plane is not detectable. The determination of the radar signature is determined by solving a matrix equation.

Without the use of the Multi Level Fast Multipole Algorithm (MLFMA), analysis of large, low-observable, structures would be impossible with full-wave methods. The interactions will be interpolated, which is translated into a more efficient matrix vector product. This matrix vector product will be solved iteratively, because a direct method would be too time consuming.

Radar signature computations are with the use of this algorithm still computationally intensive. The use of Graphical Processor Units (GPU's) can potentially improve the turn-around time. Of course, some parts of the MLFMA are better suited for this platform than other parts. Therefore, it is important to characterize the building blocks and try to find out if they are suitable to use on GPU's.

The research question for this thesis is:
Which parts of the Multi Level Fast Multipole Method could be calculated faster when those parts are transferred to a GPU, what is the expected speedup and what are the bottlenecks of using GPU's for those parts?

To answer this research question, the first chapters will give concise explanation about the Multi Level Fast Multipole Algorithm (MLFMA) and some more information about GPU computing. Afterwards the application of MLFMA on GPU's will be discussed and possible bottlenecks for this application will be identified. Finally, a mathematical design for the MLFMA on GPU clusters will be completed and the research question will be answered.

Chapter 3

Physical model for radar-cross section computations

In this thesis, calculations will be done as if there is just one radar and as if this radar sends waves of just one frequency. In real life this is much more complex with different radars sending different wave forms. The frequencies are between 1 and 50 GHz, resulting in wave lengths between 30 cm and 6 mm.
The radar in this thesis sends a transversal wave to an object. As soon as it reaches the object, the object will partly rebound this wave. The idea is to determine the equivalent current distribution induced by the radar wave on the airplane. This current distribution can then be used to determine the electromagnetic waves, reflected from the aircraft in any direction (in particular that of the radar).
Because of this radar wave, currents on one part of the aircraft induce an electric field somewhere else on the airplane.

In this thesis, it is assumed that the aircraft is made of Perfectly Electrically Conduction (PEC) materials. An advantage of this, is that there is no tangential electric field on the surface, so the magnetic current is zero. But all algorithmic computations of the MLFMA are needed to solve the scattering problem for PEC surfaces.
The results from this thesis will be the same for other materials.

For the research in this thesis, only the equations for the Electric Field Integral Equation (EFIE) will be used.

The wave forms can be derived from the Maxwell equations and the constitutive relations:

$$\nabla \times \nabla \times \boldsymbol{E} - k^2 \boldsymbol{E} = -i\omega\mu \boldsymbol{J} - \nabla \times \boldsymbol{M}, \tag{3.1}$$

where $\boldsymbol{E}$ is the electric field, $\boldsymbol{J}$ is the current density and $\boldsymbol{M}$ the magnetic charge. μ is the permeability, ω the frequency, k the wave number. $k^2 = \omega^2\mu\epsilon$ and ϵ is the permittivity of the medium in F/m.

The wave form of the Maxwell equations and the wave equation are similar, therefore solutions to the Maxwell equations can be constructed from Green's functions. [1]

If $\boldsymbol{E}^{inc}$ is the incident electric field and $\boldsymbol{H}$ the magnetic field, then for $\boldsymbol{r} \in S$ the Electric Field Integral Equation (EFIE) is given by:

$$\boldsymbol{E}^{inc}(\boldsymbol{r}) - \frac{1}{4\pi}\int_S (i\omega\mu(\boldsymbol{n}\times\boldsymbol{H})\phi - \boldsymbol{n}\times\boldsymbol{E}\times\nabla'\phi - (\boldsymbol{n}\cdot\boldsymbol{E})\nabla'\phi)d\boldsymbol{r}' = \frac{\Omega(\boldsymbol{r})}{4\pi}\boldsymbol{E}(\boldsymbol{r}). \tag{3.2}$$

In this equation, the term $\frac{\Omega(\boldsymbol{r})}{4\pi}$ finds its origin in a limit process. ∇' is the gradient on the second variable and ϕ is the Green's function and is given by:

$$\phi(\boldsymbol{r},\boldsymbol{r}') = \frac{e^{-ik|\boldsymbol{r}-\boldsymbol{r}'|}}{|\boldsymbol{r}-\boldsymbol{r}'|}. \tag{3.3}$$

The subtended solid angle $\Omega(\boldsymbol{r})$ is defined as:

$$\Omega(\boldsymbol{r}) = \lim_{h\to 0}\int_{|\boldsymbol{r}-\boldsymbol{r}'|=h,\boldsymbol{r}'\in V}\frac{1}{h^2}d\boldsymbol{r}', \tag{3.4}$$

where V is the free space surrounding the scatterer.

The surface current density $\boldsymbol{J}_S$ and the fictituous magnetic surface charge $\boldsymbol{M}_S$ are given by:

$$\boldsymbol{J}_S := \boldsymbol{n}\times\boldsymbol{H}, \tag{3.5}$$

$$\boldsymbol{M}_S := -\boldsymbol{n}\times\boldsymbol{E}. \tag{3.6}$$

As told before, the magnetic current is zero, so $\boldsymbol{M}_S = 0$.

The following equations use Maxwell equations:

$$\boldsymbol{n}\cdot\boldsymbol{E} = \frac{i}{\omega\epsilon}\nabla\cdot\boldsymbol{n}\times\boldsymbol{H} = \frac{i}{\omega\epsilon}\nabla\cdot\boldsymbol{J}_S, \tag{3.7}$$

$$\boldsymbol{n}\cdot\boldsymbol{H} = \frac{-i}{\omega\epsilon}\nabla\cdot\boldsymbol{n}\times\boldsymbol{E} = \frac{i}{\omega\mu}\nabla\cdot\boldsymbol{M}_s. \tag{3.8}$$

Using those equations, (3.2) for EFIE can be rewritten:

$$\boldsymbol{E}^{inc} - \frac{1}{4\pi}\int_S (i\omega\mu(\boldsymbol{J}_S\phi - \frac{i}{\omega\epsilon}(\nabla'\cdot\boldsymbol{J}_S)\nabla'\phi)d\boldsymbol{r}' = \frac{\Omega(\boldsymbol{r})}{4\pi}\boldsymbol{E}(\boldsymbol{r}). \tag{3.9}$$

The vector field is completely determined by the tangential part of the integral equation, because the electric current is a tangential vector field. Let $\boldsymbol{E}_t^{inc}$ be the tangential part of the incoming field. Then using $\boldsymbol{f}_t = -\boldsymbol{n}\times(\boldsymbol{n}\times\boldsymbol{f})$ for any vector field $\boldsymbol{f}$, EFIE can be written in the currents:

$$\boldsymbol{E}_t^{inc} - \frac{1}{4\pi}(\int_S i\omega\mu\boldsymbol{J}_S\phi - \frac{i}{\omega\epsilon}(\nabla'\cdot\boldsymbol{J}_S)\nabla'\phi d\boldsymbol{r}')_t = 0. \tag{3.10}$$

Remembering $k = \omega/c$ and $\epsilon\mu c^2 = 1$ gives the equations $\omega\mu = k\eta$ and $\omega\epsilon = k/\eta$. Together with the impedance $\eta = \sqrt{(\frac{\mu}{\epsilon})}$, EFIE is:

$$\boldsymbol{E}_t^{inc} - \frac{ik\eta}{4\pi}(\int_S \boldsymbol{J}_S\phi - \frac{1}{k^2}(\nabla' \cdot \boldsymbol{J}_S)\nabla'\phi d\boldsymbol{r}')_t = 0. \tag{3.11}$$

The goal is the determination of the radar cross section σ:

$$\sigma = \lim_{r\to\infty} 4\pi r^2 \frac{|\boldsymbol{E}_S|^2}{|\boldsymbol{E}^{inc}|^2}, \tag{3.12}$$

where $\boldsymbol{E}_S$ is the scattered electric field intensity, determined by the surface current density $\boldsymbol{J}_S$. The incoming signal and the signal which was sent are compared and finally the radar cross section is a measure of the detectability of the object.

Chapter 4

Method of Moments

The Method of Moments (MoM) [19] is a method that can be used to numerically solve linear partial differential equations, that are written in boundary integral form. It is a finite element discretization of an integral equation that can be written as a matrix equation.

Let $\boldsymbol{W}$ be a test vector field, which is tangential to S and δS. The weak formulation of EFIE is now obtained by multiplying (3.11) by $\boldsymbol{W}$ and integrating over S:

$$\int_S 4\pi \boldsymbol{W} \cdot \boldsymbol{E}_t^{inc} d\boldsymbol{r} = \int_S \int_S ik\eta \left(\phi \boldsymbol{W} \cdot \boldsymbol{J}_S - \frac{1}{k^2} (\nabla \cdot \boldsymbol{W}) \phi (\nabla' \cdot \boldsymbol{J}_S) \right). \tag{4.1}$$

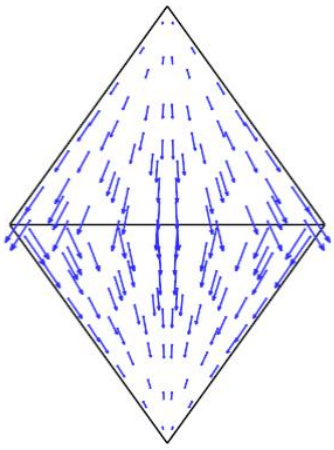


Figure 4.1: Degree of freedom or basis function living on the edge connecting two triangles.

The solution vector $\boldsymbol{J}_S$ is expanded into a linear combination of basis functions $\boldsymbol{f}_j$ (Figure 4.1), where the sum is over all internal edges:

$$\boldsymbol{J}_S = \sum_{j-1}^{N} J_j \boldsymbol{f}_j. \tag{4.2}$$

This expression, together with (4.1), gives the following expression:

$$< \boldsymbol{f}_i, 4\pi \boldsymbol{E}_t^{inc} >= \sum_j \eta \mathcal{L}_{ij} J_j, \tag{4.3}$$

where $\mathcal{L}_{ij} = \mathcal{L}(\boldsymbol{f}_i, \boldsymbol{f}_j)$:

$$\mathcal{L}_{ij} = ik \int_S \int_S (\boldsymbol{f}_i(\boldsymbol{r}) \cdot \boldsymbol{f}_j(\boldsymbol{r}') - \frac{1}{k^2} (\nabla \cdot \boldsymbol{f}_i)(\boldsymbol{r})(\nabla \cdot \boldsymbol{f}_j)(\boldsymbol{r}')) \phi(\boldsymbol{r}, \boldsymbol{r}') d\boldsymbol{r} d\boldsymbol{r}'. \tag{4.4}$$

The system matrix is full and only required for matrix vector products. For realistic radar-cross section computations, this system matrix is too large to be stored. Therefore, the Fast Multipole Method (FMM) will be discussed in the next Chapter.

Chapter 5

Multi Level Fast Multipole Method

The Multi Level Fast Multipole Algorithm (MLFMA) is a numerical method, which does not calculate the matrix, but directly solves the matrix vector product iteratively. The matrix vector products is the most time consuming part of the iterative solver. A direct method or LU decomposition will be too time consuming.
The MLFMA saves memory while increasing the computing efficiency rapidly and keeping the same precision if the integral across the globe is approached well.

As the MLFMA is an expansion of the Fast Multipole Method, the concept of the Fast Multipole Method will be explained first using one of the first papers of Rokhlin [19]. After this, the MLFMA will be introduced and the specific algorithm NLR uses (Shako) will be discussed.

5.1 Fast Multipole Method for (scalar) wave equation

In short, the Fast Multipole Method (FMM) is based on an expansion of the Green's function using a multipole expansion. The method speeds up the iterative solution of boundary-integral equations significantly. The surface scatterers or basis functions are divided into groups; basis functions that lie close together are treated as if they are a single source. Because of the use of the FMM algorithm, the complexity of the matrix vector multiplication is reduced and the far field interaction matrix does not have to be stored.
On a more abstract level, a discrete operator works on a vector with unknown coefficients of the current distribution and with the FMM the discrete operator can be simplified.

For a good understanding of the Fast Multipole Method, one of the first papers of Rokhlin is used [19]. This section is therefore closely based on the derivation presented by R. Coifman, V. Rokhlin and S. Wandzura in [19]. The equations that will be used are not exactly the same as the ones that will be used in this thesis, it is just to give a first understanding.

Rokhlin discusses a scalar problem, while the aim of this thesis is to solve a vector problem. Therefore (4.4) is in this section used in scalar form:

$$\mathcal{L}_{ij} = -i \int_S d^2\boldsymbol{r} \int_S d^2\boldsymbol{r}' f_i(\boldsymbol{r}) \frac{e^{ik|r-r'|}}{4\pi|\boldsymbol{r}-\boldsymbol{r}'|} f_j(\boldsymbol{r}'). \tag{5.1}$$

In this formula, i is the imaginary unit, $\int_S d^2\boldsymbol{r}$ are integrals over the scatterer and f_n are basisfunctions that are real and supported on local subdomains. $|r - r'|$ is the distance between two degrees of freedom and i and j are the degrees of freedom between which the interaction will be determined.

Interactions between basis functions in the system matrix $\mathcal{L}$ can be called 'near' or 'far'. This means that the matrix can be split in two matrices $\mathcal{L} = \mathcal{L}^{near} + \mathcal{L}^{far}$. $\boldsymbol{r}'$ is called near $\boldsymbol{r}$ if $|\boldsymbol{r} - \boldsymbol{r}'| < D$ and $\boldsymbol{r}'$ is called far from $\boldsymbol{r}$ if $|\boldsymbol{r} - \boldsymbol{r}'| > D$, with $\boldsymbol{r}$ and $\boldsymbol{r}'$ two different degrees of freedom and D a specific distance.

A more exact method of explaining if a degree of freedom is near of far: if a degree of freedom i is in box b, then all degrees of freedom j are near if they are also in box b, or if they are in a neighbour box of b.

If two groups are in each other's far field, the calculations can be reduced to the midpoints of the clusters instead of calculations for every pair of nodes. Something very important is the fact that if the parameters in the method for this problem are chosen right, there is almost no loss of accuracy, which makes this method a good improvement. If two clusters are not in each other's far field, the traditional MoM will be used.

The Fast Multipole Method consists of three phases: the *aggregation* phase, the *translation* phase and the *disaggregation* phase.

In the *aggregation* step, all sources are summed into a radiation pattern. This radiation pattern starts from the center of a group. From this center, outgoing waves are converted into incoming waves. This is called the *translation* step. In this step, the dense matrix can be manipulated to a diagonal matrix.

In the *disaggregation* phase, the unknown coefficients of the basis functions in a specific group are found.

The field at $\boldsymbol{r}$ from a source at $\boldsymbol{r}'$ can be computed using two displacements. The relation between the locations $\boldsymbol{r}, \boldsymbol{r}'$ and the displacement D, d are showed in the figure below. $\boldsymbol{r} - \boldsymbol{r}'$ can also be written as $\boldsymbol{r} - \boldsymbol{r}' = \boldsymbol{r} - \boldsymbol{c}_m + \boldsymbol{c}_m - \boldsymbol{c}_{m'} + \boldsymbol{c}_{m'} - \boldsymbol{r}'$ where $|\boldsymbol{r} - \boldsymbol{c}_m| = d_m$ is the distance between $\boldsymbol{r}$ and its group center $\boldsymbol{c}_m$, $\boldsymbol{c}_m - \boldsymbol{c}_{m'} = \boldsymbol{r}_{mm'}$ the distance between the group centra $\boldsymbol{c}_m$ and $\boldsymbol{c}_{m'}$ and $|\boldsymbol{c}_{m'} - \boldsymbol{r}'| = d_{m'}$ the distance between $\boldsymbol{r}'$ and its group center $\boldsymbol{c}_{m'}$. D will be close to $|\boldsymbol{r} - \boldsymbol{r}'|$, such that d is small and $d < D$. [19]
In Figure 5.1 the relations between locations and displacements can be seen, where $d = \boldsymbol{r} - \boldsymbol{c}_m + \boldsymbol{c}_{m'} - \boldsymbol{r}'$.

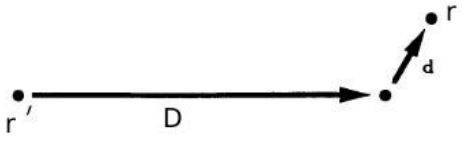


Figure 5.1: Relations between locations and displacements.

The addition theorem states the following:

$$\frac{e^{-ik|\boldsymbol{D}+\boldsymbol{d}|}}{|\boldsymbol{D}+\boldsymbol{d}|} = -ik\sum_{l=0}^{\infty}(-1)^l(2l+1)j_l(kd)h_l^{(1)}(kD)P_l(\hat{d}\cdot\hat{D}). \tag{5.2}$$

Here, j_l is a spherical Bessel function of the first kind, $h_l^{(1)}$ is a spherical Hankel function of the first kind and P_l is a Legendre polynomial.

The term $j_l P_l$ from (5.2) can also be expanded in propagating plane waves. It is nice to work with plane waves, because the waves can be split in incoming and outgoing waves: $e^{-ikd} = e^{-ik(\boldsymbol{r}-\boldsymbol{c}_m)} e^{ik(\boldsymbol{r}'-\boldsymbol{c}_{m'})}$. The term $j_l P_l$ expanded in propagating plane waves looks as follows:

$$4\pi(-i)^l j_l(kd) P_l(\hat{d}\cdot\hat{D}) = \int_{\hat{S}} e^{-ik\cdot d} P_l(\hat{k}\cdot\hat{D}), \tag{5.3}$$

where $\hat{S}$ is the unit sphere and $\hat{k}$ the different wave directions.

This expression can be substituted in (4.4) and summation and integration could be interchanged:

$$\frac{e^{-ik|\boldsymbol{D}+\boldsymbol{d}|}}{|\boldsymbol{D}+\boldsymbol{d}|} = \frac{ik}{4\pi}\int_{\hat{S}} e^{ik\cdot d}\sum_{l=0}(-i)^{l+1}(2l+1)h_l^{(1)}(kD)P_l(\hat{k}\cdot\hat{D}). \tag{5.4}$$

Calculation of an infinite sum is impossible, so the series has to be truncated. The series is divergent, so the sum should not be stopped too early, but also not too late. If this sum will be stopped at the right time, there is no problem and the error will be relatively small. Information about errors in the FMM can be found in [30].
Because of the fact that the sum is finite, it is allowed to interchange the summation and integration in (4.4).

The idea is to precompute the second part of (5.4):

$$T_L(\kappa,\cos\theta) = \sum_{l=0}^{L}(-i)^l(2l+1)h_l^{(1)}(\kappa)P_l(\cos\theta), \tag{5.5}$$

where $\kappa = kD$.

$L+1$ terms are kept; the number of terms depends on the desired accuracy and the maximum allowed value of kd.

Using this precomputed transfer operator, (5.4) can be written as follows:

$$\frac{e^{-ik|\boldsymbol{D}+\boldsymbol{d}|}}{|\boldsymbol{D}+\boldsymbol{d}|} \approx \frac{ik}{4\pi}\int_{\hat{S}} e^{ik\cdot d} T(kD,\hat{k}\cdot\hat{D}). \tag{5.6}$$

In this method, D will be chosen such that $r - r' - D$ is small. In this way, for modest values of L, the method is still accurate.
If the contribution of different degrees of freedom are combined, the FMM accelerates the calculation.

In order to give an idea of the division of the basis functions, Figure 5.2 is used. [19] To keep it easy, the small boxes in the figure consist just of one basis function. It is assumed that there are N basis functions in total, which are divided into G different groups. This means that there

are $\frac{N}{G}$ basis functions in one group m. β is called the number of the basis function in a specific group, such that (m, β) could be used as notation for all different basis functions and $n(m, \beta)$ is equal to numbers from 0 to N. The center of a group m is called c_m.

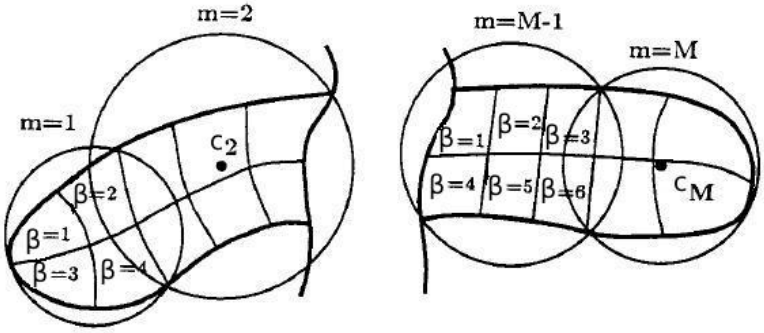


Figure 5.2: Simple example to group basis functions.

If two basis functions have regions of support that are separated by a distance not bigger than $d = \frac{1}{4}\lambda$, they are called nearby basis functions. For those group pairs with nearby basis functions, the sparse matrix $\mathcal{L}'$ will be constructed using the MoM.

For all other pairs, called far basis functions, the matrix elements are computed by the following transfer function:

$$\alpha_{ij}(\hat{k}) = \frac{k}{(4\pi)^2} \sum_{l=0}^{L} (-i)^l (2l+1) h_l^{(1)}(kD_{ij}) P_l(\hat{k} \cdot \hat{D}_{ij}), \tag{5.7}$$

where L is the number of terms in the multipole expansion.

The Fourier transforms of the basis functions are given by:

$$V_{m\beta}(\hat{k}) = \int_S d^2 r e^{ik\hat{\boldsymbol{k}}\cdot(r-c_m)} f_{n(m,\beta)}(r), \tag{5.8}$$

where $f_{n(m,\beta)}$ are scalar basisfunctions, which behave like tenac functions.

If G_m is the set of basis functions in group m and B_m all groups that are near group m, the matrix vector multiplication $\boldsymbol{\mathcal{L}J}$ can be written as:

$$\sum_{j=1}^{N} \mathcal{L}_{ij} J_j = \sum_{m' \in B_m} \sum_{j \in G_{m'}} \mathcal{L}_{ij} J_j + \int d^2\hat{k} V_{im}(\hat{k}) \sum_{m' \notin B_m} \alpha_{mm'}(\hat{k}) \sum_{j \in G_{m'}} \overline{V}_{jm'}(\hat{k}) J_j. \tag{5.9}$$

This equation is split into two parts. The first part gives the near field interations using the original MoM and the second part gives the far field interactions in terms of far fields generated by each group. $\boldsymbol{J}$ is the unknown vector.
In (5.9), the incoming and outgoing waves are independent from each other and the transfer operator is diagonal in wave space. So translation of a planar wave is a scalar operation. This means that expanding a scatterer in planar waves simplifies the interaction between different scatterers.

After a decomposition of $\mathcal{L}$, this matrix can be written as follows:

$$\mathcal{L} = \mathcal{L}' + \overline{V}^T T V, \tag{5.10}$$

where T is a matrix with the coefficients and information about the integration and $\overline{V}^T$ is the conjugate transpose of V.

5.2 The scattering problem

In Chapter 3, the physical model for EFIE is explained for PEC materials and in the previous section the concept of the Fast Multipole Method is explained using one of the first papers of Rokhlin [19]. In this chapter, the FMM for the problem of this thesis will be discussed using the notation of the national aerospace laboratory. This section is therefore closely based on the derivation presented by H. van der Ven and H. Schippers in [1].
For the research in this thesis, only the equations for the EFIE will be derived as explained in Chapter 3, therefore the unknown is a vector field.

In Chapter 4, $\mathcal{L}$ was derived in (4.4). Using the Fourier Transform, the derivatives become a product and this gives the following matrix entries $\mathcal{L}$ for the vector problem:

$$\mathcal{L}_{ij} \approx ik \int_{\hat{S}} \left(\int_S \boldsymbol{f}_i e^{-ik\hat{\boldsymbol{k}}(r-c_m)} \, \mathrm{d}\boldsymbol{r} \right) \alpha(k\hat{\boldsymbol{k}}, \boldsymbol{r}_{mm'}) (\mathbb{I} - \hat{\boldsymbol{k}}\hat{\boldsymbol{k}}) \left(\int_S \boldsymbol{f}_j e^{ik\hat{\boldsymbol{k}}\cdot(r'-c_{m'})} \, \mathrm{d}\boldsymbol{r}' \right) \, \mathrm{d}\hat{\boldsymbol{k}}, \tag{5.11}$$

where $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ are the basis functions introduced in Figure 4.1 in Chapter 4 and $\mathbb{I} - \hat{\boldsymbol{k}}\hat{\boldsymbol{k}}$ is the dyadic notation for the projection operator $\boldsymbol{v} \mapsto \boldsymbol{v} - (\hat{\boldsymbol{k}} \cdot \boldsymbol{v})\hat{\boldsymbol{k}}$. $\mathbb{I}$ comes from the basisfunctions $\boldsymbol{f}_i$ and $\boldsymbol{f}_j$ and $\hat{\boldsymbol{k}}\hat{\boldsymbol{k}}$ comes from the derivatives of the basis functions.

It is already clear to see the 3 steps of the Fast Multipole Method in this equation; the incoming waves, the transfer and the outgoing waves.

The incoming and outgoing waves, $\boldsymbol{v}_{im}^{(\mathrm{in})}(\hat{\boldsymbol{k}})$ and $\boldsymbol{v}_{jm'}^{(\mathrm{out})}(\hat{\boldsymbol{k}})$ respectively, are given by:

$$\boldsymbol{v}_{im}^{(\mathrm{in})}(\hat{\boldsymbol{k}}) = (\boldsymbol{v}_i^{(\mathrm{in})}(\hat{\boldsymbol{k}}))_m := \int_S \boldsymbol{f}_i e^{-ik\hat{\boldsymbol{k}}\cdot(r-c_m)} \, \mathrm{d}\boldsymbol{r}, \tag{5.12}$$

$$\boldsymbol{v}_{jm'}^{(\mathrm{out})}(\hat{\boldsymbol{k}}) = (\boldsymbol{v}_j^{(\mathrm{out})}(\hat{\boldsymbol{k}}))_{m'} := (\mathbb{I} - \hat{\boldsymbol{k}}\hat{\boldsymbol{k}}) \int_S \boldsymbol{f}_j e^{ik\hat{\boldsymbol{k}}\cdot(r'-c'_m)} \, \mathrm{d}\boldsymbol{r}', \tag{5.13}$$

where $\boldsymbol{v}_{im}^{(\mathrm{in})}(\hat{\boldsymbol{k}}) \in C^G$ and $\boldsymbol{v}_{jm'}^{(\mathrm{out})}(\hat{\boldsymbol{k}}) \in C^G$ are both column vectors and G is the number of groups.

In (5.11) the inproduct of $\boldsymbol{v}_{im}^{(\mathrm{in})}(\hat{\boldsymbol{k}})$ and $\boldsymbol{v}_{jm'}^{(\mathrm{out})}(\hat{\boldsymbol{k}})$ is taken, so the radial component in (5.13) does not matter and (5.12) and (5.13) are each other's complex conjugate.

The N basis functions are divided over G localized groups, each supporting about $M = N/G$ basis functions. Let G_m be the set of basis functions belonging to group m and let B_m be all groups which are near group m.

The matrix vector product $\mathcal{L}\boldsymbol{J}$ is computed as follows:

$$\sum_{j=1}^{N} \mathcal{L}_{ij} J_j = \sum_{m' \in B_m} \sum_{j \in G_{m'}} \mathcal{L}_{ij} J_j + ik \int_{\hat{S}} \boldsymbol{v}_{im}^{(\text{in})}(\hat{\boldsymbol{k}}) \cdot \left(\sum_{m' \notin B_m} \alpha_{mm'}(k\hat{\boldsymbol{k}}, \boldsymbol{r}_{mm'}) \sum_{j \in G_{m'}} \boldsymbol{v}_{jm'}^{(\text{out})}(\hat{\boldsymbol{k}}) \right) J_j \, \mathrm{d}\hat{\boldsymbol{k}}, \tag{5.14}$$

where the first part of the formula are the near field interactions and the second part the far field interactions.

The integral in (4.4) can be discretized using a quadrature rule for a sphere:

$$\int f(k) d\hat{k} = \sum_{k=1}^{K} f(k_k) w_k, \tag{5.15}$$

where w_k are the quadrature weights.

The submatrix (5.11) defines the action of degree of freedom i on degree of freedom j. The FMM transformation for all degrees of freedom i and j leads to:

$$\mathcal{L}_{ij} \rightarrow \sum_{\hat{\boldsymbol{k}}_i} (\boldsymbol{v}_{im}^{(\text{in})})^T(\hat{\boldsymbol{k}}) T_{mm'}(\hat{\boldsymbol{k}}) \boldsymbol{v}_{jm'}^{(\text{out})}(\hat{\boldsymbol{k}}) = \sum_{\hat{\boldsymbol{k}}_i} \left(\overline{\boldsymbol{v}_{im}^{(\text{out})}(\hat{\boldsymbol{k}})} \right)^T T_{mm'}(\hat{\boldsymbol{k}}) \boldsymbol{v}_{jm'}^{(\text{out})}(\hat{\boldsymbol{k}}), \tag{5.16}$$

where $T_{ij} = \alpha(k\hat{\boldsymbol{k}}, r_{mm'}) w_i \in \mathbb{C}^G$ is a dense matrix with the coefficients and information about the integration and $\left(\overline{\boldsymbol{v}_{im}^{(\text{out})}(\hat{\boldsymbol{k}})} \right)^T$ is the conjugate transpose of $\boldsymbol{v}_{im}^{(\text{out})}(\hat{\boldsymbol{k}})$. In this thesis the vectors $\boldsymbol{v}_{im}^{(\text{in})}$ and $\boldsymbol{v}_{jm'}^{(\text{out})}$ are both column vectors.

The impedance matrix $\mathcal{L}$, can then be written as follows:

$$\mathcal{L} = \mathcal{L}' + \overline{V}_{FM}^T T_{FM} V, \tag{5.17}$$

where for all the degrees of freedom (i, j) that interact using FMM $\overline{V}^T$, T_{FM} and V contain the factors $\overline{\boldsymbol{v}}_{jm'}^{(\text{out})}$, $T_{FM_{mm'}}$ and $\boldsymbol{v}_{jm'}^{(\text{out})}$ respectively. The different wave directions are included in matrices $\overline{V}^T, T_{FM}$ and V and T_{FM} is the transfer matrix for the FMM.

$\mathcal{L}'$ is the matrix with the information about the near field interactions, calculated using the original MoM method and $\overline{V}^T T_{FM} V$ the calculation of the far field interactions.

Having a closer look at (5.14) to get an idea of the dimensions of the matrices $\overline{V}^T T_{FM} V$, gives the following result:

$$\begin{aligned} \text{Tensor } \boldsymbol{v}_{jm'}^{(\text{out})}(\hat{\boldsymbol{k}}_k) = V_{m'kj} &\Rightarrow V \in \mathbb{C}^{GK \text{ x } N} \\ \text{Tensor } \alpha_{mm'}(k\hat{\boldsymbol{k}}_k, r_{mm'}) w_k = T_{FM_{mkm'k}} &\Rightarrow T_{FM} \in \mathbb{C}^{GK \text{ x } GK} \\ \text{Tensor } \boldsymbol{v}_{im}^{(\text{in})}(\hat{\boldsymbol{k}}_k) = V_{ikm} &\Rightarrow \overline{V} \in \mathbb{C}^{N \text{ x } GK} \end{aligned}$$

where K is the number of wave directions. This means that the final matrix $\mathcal{L} \in \mathbb{C}^{N \text{ x } N}$.

Having a closer look at the structure of the matrices gives the following result:

$$V = \begin{pmatrix} V_1 \\ \vdots \\ V_K \end{pmatrix},$$

where $V_1, ..., V_K \in \mathbb{C}^{G \text{ x } N}$, different for every wave direction.
And the transfer matrix T_{FM} is a block-diagonal matrix with the following structure:

$$T_{FM} = \begin{pmatrix} \boldsymbol{T}_{FM_1} & 0 & \cdots & 0 \\ 0 & \boldsymbol{T}_{FM_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{T}_{FM_K} \end{pmatrix},$$

where $\boldsymbol{T}_{FM_1}, ..., \boldsymbol{T}_{FM_K} \in \mathbb{C}^{G \text{ x } G}$, different for every wave direction.

This dense matrix $\overline{V}^T T_{FM} V$ should be multiplied by J, this will be done by first calculating VJ, afterwards multiplying T_{FM} by the result of VJ and continuing like this. The solve part is the part of the FMM that takes most of the time to calculate.

Those calculations give the following matrix dimensions:

$$\begin{aligned} (VJ) &\in \mathbb{C}^{GK \text{ x } 1} \\ (T_{FM}VJ) &\in \mathbb{C}^{GK \text{ x } 1} \\ (\overline{V}^T T_{FM}VJ) &\in \mathbb{C}^{N \text{ x } 1} \end{aligned}$$

After this specification of the matrices and the matrix vector products, it is easier to see what can be improved when performing the calculations on a GPU.

5.3 Multi Level Fast Multipole Algorithm

The Multi Level Fast Multipole Algorithm is an extension of the Fast Multipole Method. It makes use of different levels with different group-sizes, where groups or boxes are made out of smaller groups. The new terms interpolation and anterpolation are used to switch between different box sizes. This accelerates the computation speed even more than using just the FMM. In the Multi Level Fast Multipole Algorithm, a lot of calculations can be reused. For example once matrix V is known, it can be reused and does not have to be calculated again.

As explained in the previous chapter, in the FMM, $\mathcal{L}'$ is still calculated exactly. Therefore, it is preferable that $\mathcal{L}'$ contains not so many elements and therefore a fine grid is needed. For the far interactions, it is preferable to have a coarser grid, to have bigger groups and less interactions. This is why the Multi Level part is added to the FMM.

An example to understand the idea of the Fast Multipole Method a bit more, is a telephone network. There can be direct lines between all users, but there can also be hubs to connect groups of users to other groups of users so that messages could be combined. This makes the process much more efficient.

The explanation above is a very brief explanation of the multi level part of the MLFMA. To give a better idea of how the method works, Figure 5.3 shows a schematic example of the MLFMA. [1]

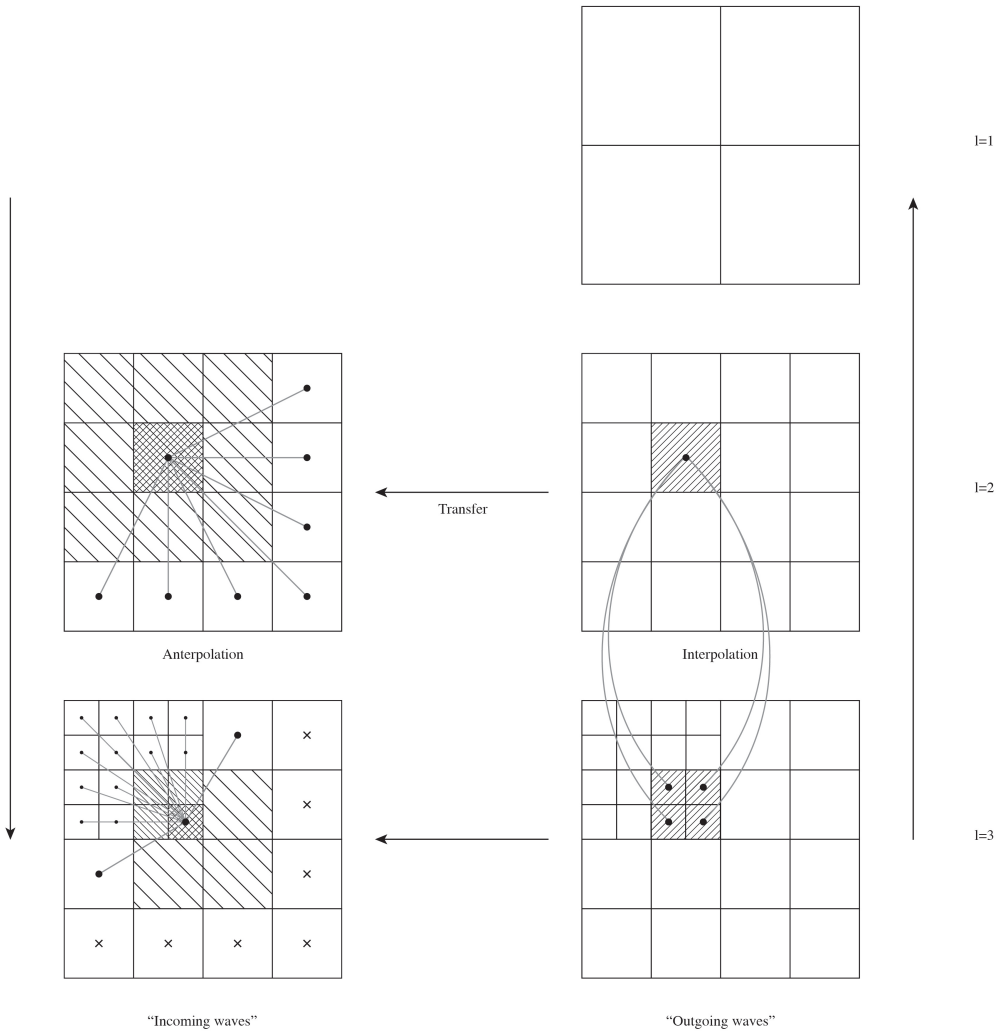


Figure 5.3: Schematic example of the MLFMA.

The finest level contains the smallest cubes in the octree and they contain at most a few degrees of freedom. In a coarser level, one cube contains eight fine cubes. Continuing like this, the coarsest level just consists of a single cube, encompassing the complete scatterer.
Figure 5.3 shows the outline of the MLFMA. The idea is to start with the outgoing waves on the level of the degrees of freedom and interpolate (agglomeration) this to clusters until the level $l = 2$. At this level it is not possible to go to bigger clusters, because then there would not be far field interactions at all. Later in this section it will be explained why.
The transfer operator transfers the outgoing waves at each level. Then anterpolation changes the clusters back to the level of degrees of freedom and then it is translated into an incoming wave.
The distance between the basis functions is important for the level on which the interactions are calculated: interactions between scatterers at large distance are computed at a coarse level, while interactions between scatterers at a smaller distance are computed on a finer level.

In the anterpolation phase, at each level the far field interactions are calculated. The interaction with all degrees of freedom that are neighbours of neighbours are calculated at that level. At a finer level, those far field interactions do not have to be calculated again.

Having the schematic representation of the MLFMA of Figure 5.3 in mind, the MLFMA will now be explained a bit more in detail.
At the finest level $l = L$ the outgoing waves $\boldsymbol{v}^{(0)}$ and $\boldsymbol{w}^{(0)}$ for all group centers at this level are calculated. After this calculation, the outgoing waves are interpolated to the group centers at the levels $l = L - 1, ..., 2$. At each level, starting from the coarsest level $l = 2$, the interactions between the groups at this level are computed. Now, at each level the incoming waves computed before will be inversely interpolated (or anterpolated) to level $l + 1$ and added to the incoming waves computed directly in this level. At the finest level $l = L$, the near interactions will be computed using the MoM.

In the previous section about the FMM it is explained that the geometric size determines the accuracy of which the integral over the unit sphere should be determined. For the MLFMA, the interpolation is needed because the group diameter increases with coarser levels and therefore the number of plane wave directions required for an accurate solution increases as well.

As explained in the previous section, at a specific level, only the interactions between degrees of freedom that are neighbours of neighbours are calculated. This means that for a given group m at level l, only the interactions with groups that are children of the neighbours of the parent of group m, which are not a neighbour of m. In this way, the interactions are computed in the most efficient way. Other groups at this level are further away from group m and can be computed at a coarser level.
As can be seen in Figure 5.3, $l = 2$ is the coarsest active level. This is because level 1 consists of eight cubes which are neighbours, so the FMM can not be applied at this level.

At all finer levels, the incoming waves consist of the interactions at this level and the incoming waves projected from the next coarser level. The anterpolation step is the transpose of the interpolation step.
At the finest level $l = L$, the incoming waves are combined with the near interactions computed with the MoM.

During the interpolation step, the plane wave expansions from each child to its parent will be interpolated, then the interpolated plane waves will be translated from the center of the child

to the center of the parent and finally the plane waves over all children will be summed.
The translation step consists of a premultiplication for a given wave direction $\hat{\boldsymbol{k}}$:

$$e^{-ik\hat{\boldsymbol{k}}(\boldsymbol{c}_l^i - \boldsymbol{c}_{l-1}^j)}, \tag{5.18}$$

where $\boldsymbol{c}_{l-1}^j$ is the group center of the j-th box at level $l-1$, parent box b_{l-1}^j, and box b_l^i at level l is one of the children of box b_{l-1}^j.

The interpolation step consists of Lagrange interpolation of the plane wave expansions, translation of the interpolated plane waves and the summation over all plane waves.

For a given set of data points $\{g_k = g(x_k) | 0 \leq k \leq N\}$, the Lagrange interpolated function $\tilde{g}$ is defined as:

$$\tilde{g}(x) = \sum_{k=0}^{N} l_k(x) g_k, \tag{5.19}$$

where the interpolators l_k are defined as:

$$l_k(x) = \Pi_{i=0, i\neq k}^{N} \frac{x - x_i}{x_k - x_i}. \tag{5.20}$$

The problem is two dimensional, but this will be split in two one dimensional problems with spherical coordinates θ and ϕ.

The interpolator is the same for all groups at a given level, so the interpolation matrices for all levels can be precomputed and stored.

The group-to-group interactions between two groups m and m' at a given level consists of the premultiplication with $\alpha(k\hat{\boldsymbol{k}}, \boldsymbol{c}_m - \boldsymbol{c}_{m'})$ for a given wave direction $\hat{\boldsymbol{k}}$. For this given wave direction, the transfer operator only depends on the distance vector $\boldsymbol{c}_m - \boldsymbol{c}_{m'}$.
The influence list consists at most of all boxes contained in the three box layers around the given box: $(3+1+3)^3 = 7^3$ boxes. The box itself and its neighbours are excluded from the influence list and therefore the maximum number of possible paths $\boldsymbol{c}_m - \boldsymbol{c}_{m'}$ is equal to $7^3 - 27 = 316$.

For the MLFMA, the algorithm can be given in five steps. First, at the finest level, the outgoing waves are calculated as before:

$$\boldsymbol{v}_{m,l_{max}}^{(\text{out})}(\hat{\boldsymbol{k}}_i^{l_{max}}) := \sum_{j \in G_m^{l_{max}}} (\mathbb{I} - \hat{\boldsymbol{k}}_i^{l_{max}} \hat{\boldsymbol{k}}_i^{l_{max}}) \int_S \boldsymbol{f}_j e^{ik\hat{\boldsymbol{k}}_i^{l_{max}} \cdot (\boldsymbol{r} - \boldsymbol{c}_m^{l_{max}})} d\boldsymbol{r} J_j, \tag{5.21}$$

where $m = 1, .., N^{l_{max}}$, with $N^{l_{max}}$, is the number of groups at level l and $i = 1, ..., K_{l_{max}}$, with $K_{l_{max}}$, is the number of integration points used in the k-space integral at level l.

The aggregation from fine to coarser levels for $l = l_{max}, ..., 2$ is given as follows:

$$\boldsymbol{v}_{m,l}^{(\text{out})}(\hat{\boldsymbol{k}}_j^l) := \sum_{m_c \in G_m^l} e^{ik\hat{\boldsymbol{k}}_j^l \cdot (c_{m_c}^{l+1} - c_m^l)} \tilde{\boldsymbol{v}}_{m_c,l+1}^{(\text{out})}(\hat{\boldsymbol{k}}_j^l), \tag{5.22}$$

where $m = 1, ..., N^l$, with N^l the number of groups at level l, $j = 1, ..., K_l$ the directions at the coarser levels and $\tilde{\boldsymbol{v}}_{m_c,l+1}^{(\text{out})}(\hat{\boldsymbol{k}})$ the interpolated far fields at the finer level $\boldsymbol{v}_{m_c,l+1}^{(\text{out})}(\hat{\boldsymbol{k}}_i^{l+1})$, where m_c is the center of one of the child boxes c.

After this aggregation step, the far field radiation patterns will be transferred to the groups in the influence list at each level as follows:

$$\boldsymbol{v}_{m_2,l}^{(\text{out})}(\hat{\boldsymbol{k}}_i^l) := \sum_{m_1 \in G_{m_2}^{int,l}} w_i^l \alpha(k\hat{\boldsymbol{k}}_i^l, \boldsymbol{c}_{m_2}^l - \boldsymbol{c}_{m_1}^l) \boldsymbol{v}_{m_1,l}^{(\text{out})}(\hat{\boldsymbol{k}}_i^l), \tag{5.23}$$

where $m_2 = 1, ..., N^l$ and $l = l_{max}, .., 2$.

The incoming waves should be disaggregated from the coarser levels to the finest level:

$$\boldsymbol{v}_{m,l}^{(\text{out})}(\hat{\boldsymbol{k}}_j^l) := e^{ik\hat{\boldsymbol{k}}_j^l \cdot (\boldsymbol{c}_{m_p}^{l-1} - \boldsymbol{c}_m^l)} \tilde{\boldsymbol{v}}_{m_p,l-1}^{(\text{out})}(\hat{\boldsymbol{k}}_j^l) + \boldsymbol{v}_{m,l}^{(\text{out})}(\hat{\boldsymbol{k}}_j^l), \tag{5.24}$$

where $m_p = 1, ..., N^{l-1}$ and $\tilde{\boldsymbol{v}}_{m_p,l-1}^{(\text{out})}$, $m \in m_p$ the interpolated incoming waves.

Now, the final incoming wave can be computed:

$$O_j := ik \sum_{i=1}^{K_{l_{max}}} (\mathbb{I} - \hat{\boldsymbol{k}}_i^{l_{max}} \hat{\boldsymbol{k}}_i^{l_{max}}) \int_S \boldsymbol{f}_j e^{ik\hat{\boldsymbol{k}}_i^{l_{max}} \cdot (\boldsymbol{r} - \boldsymbol{c}_i^{l_{max}})} d\boldsymbol{r} \cdot \boldsymbol{v}_{m,l_{max}}^{(\text{out})}(\hat{\boldsymbol{k}}_i^{l_{max}}), \tag{5.25}$$

where $j = 1, ..., N$.

The name of the outgoing wave changes in an incoming wave, when the wave arrived in the group center of the degree of freedom on the finest level.

To give a better overview of the five steps of the MLFMA, the impedance matrix $\mathcal{L}$ can be written as matrix multiplications:

$$\begin{aligned}
\mathcal{L} &= \mathcal{L}' + \overline{V}^T T_{l_{max}} V + \overline{V}^T A_{l_{max}-1}^{l_{max}T} T_{l_{max}-1} A_{l_{max}}^{l_{max}-1} V + ... \\
&= \mathcal{L}' + \overline{V}^T (T_{l_{max}} + A_{l_{max}-1}^{l_{max}T} T_{l_{max}-1} A_{l_{max}}^{l_{max}-1}) V + ... \\
&= \mathcal{L}' + \overline{V}^T (T_{l_{max}} + A_{l_{max}-1}^{l_{max}T} (T_{l_{max}-1} + A_{l_{max}-2}^{l_{max}-1T} T_{l_{max}-2} A_{l_{max}-1}^{l_{max}-2}) A_{l_{max}}^{l_{max}-1}) V + ... \\
&= \mathcal{L}' + \overline{V}^T (T_{l_{max}} + A_{l_{max}-1}^{l_{max}T} (T_{l_{max}-1} + ...(... + A_{l=2}^{l=3T} T_{l=2} A_{l=3}^{l=2})...) A_{l_{max}}^{l_{max}-1}) V
\end{aligned} \tag{5.26}$$

where matrix A is the agglomeration step, so this matrix contains the information about the interpolation and anterpolation.

To get an idea of the dimensions of the matrices in (5.26), the structure of each matrix is given containing the different wave directions.

$$
\begin{aligned}
\text{Tensor } \boldsymbol{v}^{(\text{out})}_{m,l_{max}}(\hat{\boldsymbol{k}}_i^{l_{max}}), \forall \hat{\boldsymbol{k}}^{l_{max}} = V_{m'j} &\Rightarrow V \in \mathbb{C}^{G_{l_{max}} K_{l_{max}} \text{ x } N} \\
\text{Tensor } \boldsymbol{v}^{(\text{out})}_{mj,l_{max}-1} = A^{l_{max}-1}_{l_{max}} &\Rightarrow A^{l_{max}-1}_{l_{max}} \in \mathbb{C}^{G_{l_{max}-1} K_{l_{max}-1} \text{ x } G_{l_{max}} K_{l_{max}}} \\
&\vdots \\
\text{Tensor } \boldsymbol{v}^{(\text{out})}_{mj,l=2} = A^{l=2}_{l=3} &\Rightarrow A^{l=2}_{l=3} \in \mathbb{C}^{G_{l=2} K_{l=2} \text{ x } G_{l=3} K_{l=3}} \\
\text{Tensor } \boldsymbol{v}^{(\text{out})}_{m,l=2}(\hat{\boldsymbol{k}}_i^{l=2}), \forall \hat{\boldsymbol{k}}^{l=2} = T_{l=2} &\Rightarrow T_{l=2} \in \mathbb{C}^{G_{l=2} K_{l=2} \text{ x } G_{l=2} K_{l=2}} \\
&\vdots \\
\text{Tensor } \boldsymbol{v}^{(\text{out})}_{m,l_{max}}(\hat{\boldsymbol{k}}_i^{l_{max}}), \forall \hat{\boldsymbol{k}}^{l_{max}} = T_{l_{max}} &\Rightarrow T_{l_{max}} \in \mathbb{C}^{G_{l_{max}} K_{l_{maX}} \text{ x } G_{l_{max}} K_{l_{max}}} \\
\text{Tensor } \boldsymbol{v}^{(\text{out})}_{mj,l=2} = A^{l=3}_{l=2} &\Rightarrow A^{l=3}_{l=2} \in \mathbb{C}^{G_{l=3} K_{l=3} \text{ x } G_{l=2} K_{l=2}} \\
&\vdots \\
\text{Tensor } \boldsymbol{v}^{(\text{out})}_{mj,l_{max}-1} = A^{l_{max}}_{l_{max}-1} &\Rightarrow A^{l_{max}}_{l_{max}-1} \in \mathbb{C}^{G_{l_{max}} K_{l_{max}} \text{ x } G_{l_{max}-1} K_{l_{max}-1}} \\
\text{Tensor } \boldsymbol{v}^{(\text{in})}_{m,l_{max}}(\hat{\boldsymbol{k}}_i^{l_{max}}), \forall \hat{\boldsymbol{k}}^{l_{max}} = V_{im} &\Rightarrow V \in \mathbb{C}^{N \text{ x } G_{l_{max}} K_{l_{max}}}
\end{aligned}
$$

This means that the final matrix $\mathcal{L} \in \mathbb{C}^{N \text{ x } N}$.

Matrices V and T have the same structure as explained in the chapter about the FMM method. The agglomeration matrix A^l_{l-1} is constructed in the same way as matrix A^{l-1}_l; calculation of the Lagrange interpolation matrix $I^l_{l-1} \in \mathbb{R}^{G_l K_l \text{ x } G_l K_{l-1}}$ and afterwards the phase shift and sum over all child groups. The Lagrange interpolation matrix I^l_{l-1} looks as follows:

$$
I^l_{l-1} = \begin{pmatrix}
\mathcal{I}^l_{l-1} & 0 & \cdots & 0 \\
0 & \mathcal{I}^l_{l-1} & \ddots & \vdots \\
\vdots & \ddots & \ddots & 0 \\
0 & \cdots & 0 & \mathcal{I}^l_{l-1}
\end{pmatrix},
$$

where in this case $\mathcal{I}^l_{l-1} \in \mathbb{R}^{K_l \text{ x } K_{l-1}}$ and the matrix is the same for each group.
After the phase shift and the sum, matrix $A^l_{l-1} \in \mathbb{C}^{G_l K_l \text{ x } G_{l-1} K_{l-1}}$ is a obtained.

The agglomeration matrix A^{l-1}_l is the transpose of matrix A^l_{l-1} and I^{l-1}_l is the transpose of matrix I^l_{l-1}.
The final matrix I^{l-1}_l looks as follows:

$$
I^{l-1}_l = \begin{pmatrix}
\mathcal{I}^{l-1}_l & 0 & \cdots & 0 \\
0 & \mathcal{I}^{l-1}_l & \ddots & \vdots \\
\vdots & \ddots & \ddots & 0 \\
0 & \cdots & 0 & \mathcal{I}^{l-1}_l
\end{pmatrix},
$$

where in this case $\mathcal{I}^{l-1}_l \in \mathbb{R}^{K_{l-1} \text{ x } K_l}$ and the matrix is the same for each group.
Finally, matrix $A^{l-1}_l \in \mathbb{C}^{G_{l-1} K_{l-1} \text{ x } G_l K_l}$ is obtained.

The matrix for the far interactions in the second part of (5.26) should be multiplied by J, this will be done by first calculating VJ, afterwards multiplying $A_{l_{max}}^{l_{max}-1}$ by the result of VJ and continuing like this.

Those calculations give the following matrix dimensions:

$$\begin{aligned}
(VJ) &\in \mathbb{C}^{G_{l_{max}} K_{l_{max}} \text{ x } 1} \\
(A_{l_{max}}^{l_{max}-1} VJ) &\in \mathbb{C}^{G_{l_{max}-1} K_{l_{max}-1} \text{ x } 1} \\
&\vdots \\
(A_{l=4}^{l=3} ... A_{l_{max}}^{l_{max}-1} VJ) &\in \mathbb{C}^{G_{l=3} K_{l=3} \text{ x } 1} \\
((T_{l=3} + A_{l=2}^{l=3^T} T_{l=2} A_{l=3}^{l=2}) A_{l=4}^{l=3} ... A_{l_{max}}^{l_{max}-1} VJ) &\in \mathbb{C}^{G_{l=3} K_{l=3} \text{ x } 1} \\
(A_{l=3}^{l=4^T} (T_{l=3} + A_{l=2}^{l=3^T} T_{l=2} A_{l=3}^{l=2}) A_{l=4}^{l=3} ... A_{l_{max}}^{l_{max}-1} VJ) &\in \mathbb{C}^{G_{l=4} K_{l=4} \text{ x } 1} \\
&\vdots \\
(A_{l_{max}-1}^{l_{max}^T} ... A_{l=3}^{l=4^T} (T_{l=3} + A_{l=2}^{l=3^T} T_{l=2} A_{l=3}^{l=2}) A_{l=4}^{l=3} ... A_{l_{max}}^{l_{max}-1} VJ) &\in \mathbb{C}^{G_{l_{max}} K_{l_{max}} \text{ x } 1} \\
(\overline{V}^T A_{l_{max}-1}^{l_{max}^T} ... A_{l=3}^{l=4^T} (T_{l=3} + A_{l=2}^{l=3^T} T_{l=2} A_{l=3}^{l=2}) A_{l=4}^{l=3} ... A_{l_{max}}^{l_{max}-1} VJ) &\in \mathbb{C}^{N \text{ x } 1}
\end{aligned}$$

After this specification of the matrices and the matrix vector products, it is easier to see what can be improved when performing the calculations on a GPU.

5.4 Linear solver

The NLR in-house developed algorithm for high-frequency scattering analysis for very large objects can be used. This algorithm solves the resulting linear system using the Generalized Minimal Residual (GMRES) solver combined with a preconditioner based on the near interaction matrix. EFIE with the same testfunctions and basisfunctions obtains a symmetric matrix. For complex and symmetric matrices, too little information is known to use a more precise solver; the matrix should have been hermition. More information about the algorithm can be found in [1].

GMRES is used for solving large sparse linear systems. It minimizes the residual over the Krylov subspace span$\{r_0, Ar_0, A^2 r_0, ..., A^i r_0\}$, with $r_0 = b - Ax_0$.
The advantages of a GMRES solver are its excellent convergence characteristics and the fact that the memory use only depends on the size of the Krylov space in the GMRES solver.
An important problem of the GMRES solver is the fact that the basis for the Krylov subspace must be stored in the iterative process, which demands large memory space. Because of the fact that it is allowed to work with nonsymmetrical matrices, each new vector has to be orthogonalized against all previously generated basis vectors, which results in long recurrences.

An option to avoid huge memory storage is to restart the GMRES method at each m iterations, where m is very small and x_m the initial guess to the next iteration. In this case the size of the basis is m orthogonal vectors. This will probably not speedup the speed of convergence, because the GMRES method will start again, but in this case with a starting vector which is a bit closer to the optimal solution. Probably, this will not be a good idea, because this means the used preconditioner is not good enough.
A suitable preconditioner can reduce the number of Krylov iterations.

The algorithm in this thesis uses very large matrices and therefore it probably will not be a problem to save the whole Krylov basis. This will take very little space compared with the matrices that have to be stored.

5.5 Overview building blocks MLFMA

In the first sections of this chapter, it is explained that the MLFMA is a method which splits the interactions in two parts: far and near field interactions. The near interaction matrix is still calculated using the original MoM method. For the far interactions, a few matrix multiplications should be done.

The building blocks of the MLFMA consist of four different components, which will be discussed in more detail below.

First of all, the operators for the MLFMA should be determined. This means that matrices T_l, V, $\overline{V}$, A_{l-1}^{l} and A_{l}^{l-1} should be constructed. Matrix $T_l \in \mathbb{C}^{G_l K_l \text{ x } G_l K_l}$ is a sparse block-diagonal matrix that contains information for different wavelengths and contains information about the transfer. To construct this matrix, many Hankel functions and Legendre polynomials should be calculated. Matrix $V \in \mathbb{C}^{N \text{ x } G_l K_l}$ is a dense matrix that contains information about the incoming or outgoing waves and has to calculate many exponential functions. Interpolation (or anterpolation) matrix $A_{l}^{l-1} \in \mathbb{C}^{G_{l-1} K_{l-1} \text{ x } G_l K_l}$ is a sparse matrix that contains information about the agglomeration. Again, a lot of exponential functions have to be calculated to construct this matrix.
The construction of those matrices is parallelizable, as they are not inherently sequential.

Secondly, the near field matrix $\mathcal{L}'$ should be constructed. This matrix vector product is also parallelizable; this is used a lot already in practice.

The construction of the preconditioner can also be done in parallel as soon as the near field matrix is known. The preconditioner is a sparse matrix.

The solve of the linear system using GMRES consists of two phases: the matrix vector product and the construction of the basis of the Krylov space. The matrix vector product can be parallelized and the construction of the Krylov basis is inherently sequential, so cannot be parallelized as a whole.

5.6 Message Passing Interface parallelization of MLFMA

The National Aerospace Laboratory uses a Message Passing Interface (MPI) parallelization method for the MLFMA. This means parallelizing the method on multiple CPU's. In the next Chapter, the differences between MPI's and Graphics Processing Units (GPU's) will be explained.

To parallelize an algorithm, data should be divided over different Central Processing Units (CPU's). A problem that arised when started working with the MPI parallelization method, is the fact that the MLFMA sometimes performs a task for different groups and sometimes for different wave directions. This means that a choice should be made if the groups or the wave

directions are parallelized. In both ways data transfer is needed, because not all data is present in the memory of a specific CPU.

The first idea was the parallelization over the wave directions. This means that during the agglomeration, data should be transfered, while the transfer part can be done very efficient because the transfer matrix is dependent of the wave directions. The agglomeration with data transfer was working well on coarse levels, but on finer levels a lot of data had to be transferred. On the finest levels, there are very little wave directions, so there is not a lot to divide on the GPU. Therefore the second idea was to parallelize over the groups at the finest levels and parallelize over the wave directions on coarser levels. This means that there should be a border between the two different ways of parallelizing, where all CPU's should communicate with each other.
This second idea was working much faster than the first one. Hopefully working with Graphics Processing Units (GPU's) will make the algorithm even faster.

Chapter 6

Graphics Processing Units

6.1 Background of Graphics Processing Units

Graphics Processing Units (GPU's) can be used for parallel computing. Without parallel computing, a specific task can only start if the previous task has finished. On parallel computers more than one task can be done at the same time, which could speedup the rate of performance enormously for very large problems. Supercomputer performance can be obtained for relatively low costs. One has to do some work to make a specific algorithm suitable for the GPU.
For this thesis, CUDA (Compute Unified Device Architecture) from NVIDIA will be used and the programming language will be C.
A GPU is a Single Instruction Multiple Data (SIMD) computer, this means that multiple processors perform the exact same instruction on multiple elements.

A Graphics Processing Unit (GPU) takes over tasks of a Central Processor Unit (CPU). Calculations will be done much faster on a GPU because it consists of thousands of small, efficient cores, while a CPU consists only of a few cores. The most intensive functions can be computed by a GPU, while other calculations can still be performed by the CPU.

In GPU's, different cores could perform a calculation at the same time. Those different cores should perform the same operation for different data, so for example perform a multiplication with different input variables.

GPU's can only be used for 'inner-loop' style codes, but different inner-loops can be divided over different blocks so that at the end an outer-loop can be calculated (partly) on a GPU.

6.2 Threads, blocks and grids

A GPU is made up of threads, blocks and grids. *Threads* calculate the same parallel computations that should be performed, but on different parts of an array. A group of threads together is called a *block* and a group of blocks together is called a *grid*. One entire grid is handled by a single GPU chip. This chip is organized as a collection of *multiprocessors* (MPs) and those MPs are responsible for handling one or more blocks. It never happens that a block is divided over multiple MPs.

Every MP is then again subdivided into multiple *stream processors* (SPs) and those SPs are responsible for one or more threads.

The overview of the threads, blocks and grids can be seen in Figure 6.1.

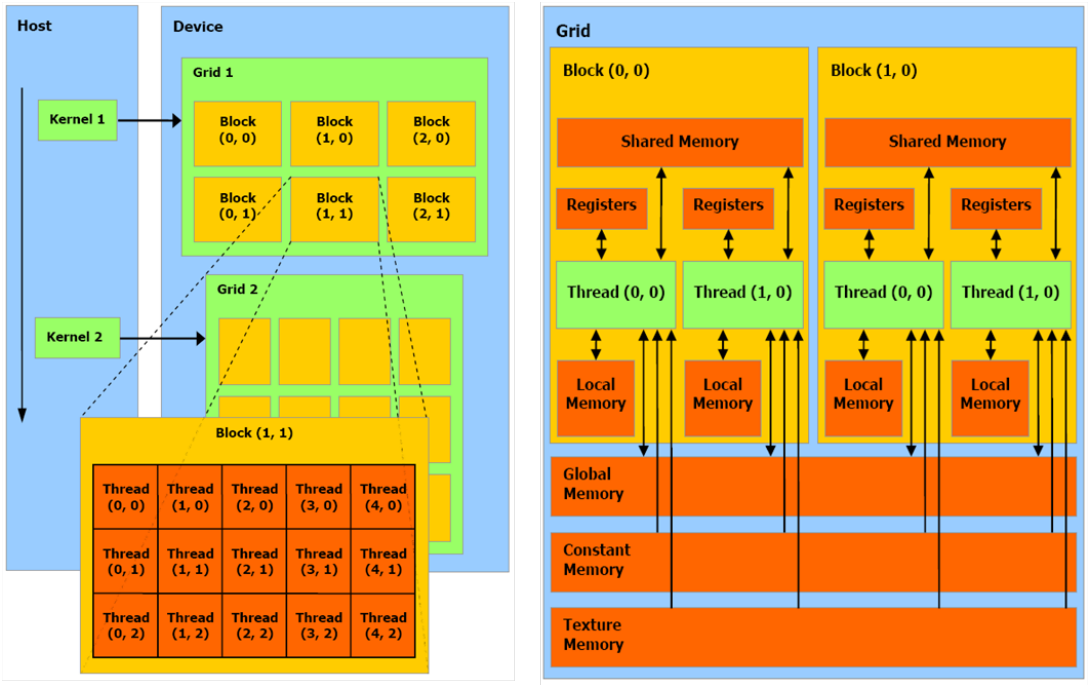


Figure 6.1: Overview of GPU architecture. [31]

6.3 Types of memory

There are different memory spaces on a GPU and they all have advantages and disadvantages. Therefore it is important to make use of the right memory. Global, local, texture, constant, shared and register memory are different places to save memory, which will be briefly explained below. Next to the speed of the memory, the scope and the lifetime of the memory are also important to keep in mind.

Register memory and local memory both last for the lifetime of the thread that wrote the data. The written data is only visible to this thread. The difference between the two memories is that the local memory performs slower than the register memory. If threads would like to communicate with each other, threads need to send and receive data. This communication needs an interconnection network via memory.
Shared memory makes it possible for threads within an MP to communicate and share data between each other. The shared memory lasts for the duration of the block and the data is visible to all threads in this block. It could happen that two processors read and write information on the same part of the memory, therefore the order of the processes is very important.
Global memory also makes it possible for threads to communicate, but this memory is visible to all threads within the application and lasts for the duration of the host allocation. Global memory is also called device memory. In a multiprocessor, every thread can read and write to any location in the memory.

Constant memory and texture memory are both read only and therefore beneficial for very special types of applications. Using one of those types of memories can reduce the required memory bandwith when threads read the same location.
The overview of the different types of memory on a GPU can be seen in Figure 6.1.
On a GPU, shared memory is the type of memory which is used the most.

The cache is a part of the memory between the main memory and the vector registers. A vector register works on a vector processor and contains vectors. Data transfer between cache and vector is fast, but data transfer between cache and main memory is slow.

Moving n data items to another location depents on the latency or start-up time α and the incremental time per data item moved: β. Therefore a simple formula for the time it takes to move n data items is $\alpha + \beta n$.
If a computer has a relatively high latency, it is useful to combine communications.

In this section, different types of memory are discussed. Important is that there is very little memory available on a GPU.

6.4 Data transfer

For data transfer between system memory and graphic's card memory and vice versa, pinned memory or non-pinned memory can be used. Pinned memory provides improved transfer speeds, but is much more expensive to allocate and deallocate compared to non-pinned memory. This means pinned memory is useful if a lot of memory needs to be transferred, because then it provides a performance advantage.

6.5 Single and double precision processing

Both single and double precision floating-point formats are used on GPU's. The single precision format occupies 4 bytes (32 bits) in computer memory and could be used for computational expensive parts of the calculations. The double precision format occupies 8 bytes (64 bits) in computer memory and is preferred to have a lower aggregate error but is also slower than the single precision format.
The difference between the price of a GPU with or without double precision is big.

On current CPU's single and double precision are equally fast, while on a GPU this can differ a factor 2 to 8.
A solution for the fact that double precision is more time consuming is combining single and double precision. In this case, with little extra work a double precision result is obtained with the amount of work of single precision.

For the algorithm used in this thesis, single precision is enough.

6.6 Clusters of Graphics Processing Units

There are two types of clusters, it could be 1 CPU with more GPU's or a cluster of CPU's with each a GPU. The workload can in those cases be spread over the different GPU's, which makes

the calculations go faster by making use of the right configuration.

For simplicity, clusters will not be used in this thesis. But if the acceleration is good, clusters can be used to make the acceleration even better.

6.7 Message Passing Interface versus Graphics Processing Units

MPI is coarse parallel, so different calculations can be performed by different computers. If MPI is used, all the threads are independent and therefore need to communicate by sending messages. MPI can be used in C and Fortran and contains a rich set of routines.
GPU is fine parallel, so just exactly the same orders can be performed on the nodes. Therefore a structured way of memory use is desirable. In a GPU, threads can communicate with each other if the right memory is used. Remember that this memory is limited, so inifinitely big problems can not be solved by just one GPU. If in one problem a lot of the same work has to be done, GPU's are convenient to use.

6.8 Vector Computers versus Parallel Computers

Vector Computers were used to parallelize computations before the invention of GPU's. It is very important to perform the loops and memory in a structured way for a Vector Computer. Vector operations are performed fast, but different structures are performed slower.
Vector Computers contain vector registers, which can be used to store intermediate and final results. The vector registers are very costly, so only a small number of reals can be stored. If a vector is too large to store in the vector registers, the Vector Computer evaluates the calculation in more parts.

Vector Computers are mainly focussed on performing vector operations and perform slower when working with different structures. For GPU's this difference does not matter that much. It is also less important to perform the loops and memory in a structured way on a GPU.

6.9 Common pitfalls and tips for Graphics Processing Units

GPU's are already used a lot in practice to perform matrix vector multiplications. The main focus of this thesis is the matrix vector solve in the MLFMA. It will be checked if, and how this could be done efficiently by GPU's and what the bottlenecks are.

It is important to know what happens exactly when working with GPU's. It could happen that algorithms will only become slower when running them on GPU's and sometimes it is needed to do more work than on a CPU to achieve a faster calculation on a GPU.

A GPU can perform multiple calculations at the same time, but it is important to keep in mind those calculations can just be simple calculations. In case of the MLFMA, the algorithm should be split into a lot of smaller calculations which can be performed on a GPU.
Sometimes it is needed to perform more steps or more calculations than on a CPU, but at the end it is still faster if it is performed on a GPU.

One of the causes of a longer computing time on a GPU is the fact that sending information from the CPU to the GPU, and the other way around, is time-consuming. It could happen that sending the information costs more time than the actual speedup using a GPU. Therefore, it is important to understand exactly what happens and for example send a lot of data at the same time, so that the time to send this information is negligible.

Like already explained before, the GPU is made to perform a lot of the same calculations. If only two of the same calculations should be performed, it does not really make sense to use a GPU. In this case sending data would probably be more time consuming than the calculation itself.

Another important thing to check is the fact if there is a recursion in the algorithm. Matrix vector products can be done perfectly in parallel, but if there is a recursion in the algorithm, calculations can not be done independently of each other and can therefore not be done in parallel. An example of a recursion which can not be done in parallel is the following:

do $1 = 1, 10$
  $y(i) = b(i) - a(i)y(i-1)$
enddo

In the previous chapter the GMRES method was explained very briefly. Quite some inner products should be calculated for this method and all those inner products need global communication. It could therefore happen that another, slower, solver works faster on a GPU.

The operators $+$, $-$ and $*$ all count as one flop, while the division operator is much more expensive. Therefore it is important to avoid this operator as much as possible. It is for example possible to use the operator one time ($h = 1/b$) and afterwards multiply the rest by h, which gives the same result.

Finally, the most important thing is the amount of data transfer. If too much data should be transferred, the method will not work faster on a GPU. The implementation strategy with as less data transfer as possible is the best one.

6.10 Kepler K20

For this thesis, a CUDA 4230 GPU Supercomputer, KEPLER ready with an NVIDIA TESLA Kepler K20 GPU Computing Card and Intel Xeon X5650 will be used. Specifications of the K20 can be found in the table below, where the Streaming Multiprocessor (SM) is the part on the GPU where processing elements are grouped together and the instructions are stored. [34]

Released	November 2012
Processing elements	2496
Streaming mult. processors	13
Processor clock (MHz)	705.5
Memory clock (MHz)	5200
Memory bandwidth (GB/s)	208
Memory bus interface PCIe	2.0x16
Max. global memory (MB)	5120
Shared memory (per SM)	48 kB
Registers (per SM)	256 kB
Double-precision GFLOPS	1173

Chapter 7

The application of MLFMA on GPU clusters

There are three parts of the MLFMA which can potentially speedup the process by parallel computing. Namely, the construction of the matrices V, A and T, the construction of the near field system matrix $\mathcal{L}'$ and the matrix vector multiplication in the solve part where the matrices are multiplied by the unknown vector J.

7.1 First ideas

In the final section of the Multi Level Fast Multipole Algorithm, the building blocks for the MLFMA are discussed. Having a closer look at those building blocks and getting a first idea of the possibility of using GPU's in different parts of the calculation without reading the literature, gives the following outcomes.

The construction of the near field system matrix $\mathcal{L}'$ can be done faster on GPU's by letting each thread calculate one matrix entry. This is already done a lot in practice, but this is not the most time consuming part of the whole algorithm. Therefore this is something which will be implemented potentially at the end.

The fill of the matrices V, T and A can probably be calculated faster with the use of GPU's. Because of the use of e-powers, Hankel functions and Legendre polynomials, which are time-consuming calculations, the fill could be done faster using GPU's by letting each thread calculate one matrix entry.
It is important to always check if a speedup is obtained, because it could happen that sending the data is more time-consuming than the time saved by using GPU's.

For the interpolation step, the NLR makes use of Lagrange interpolation, which is local interpolation because only information of the neighbours is needed. It could be checked if global interpolation works more efficient on GPU's. Probably the local interpolation method will still be faster on a GPU, but it could happen that this is not the case and global interpolations make the calculations a lot faster on a GPU.

The challenge in parallelizing the MLFMA is the fact that there are two possible ways of parallelizing: over the groups or over the wave directions. Both of them have an advantage at a

specific part of the MLFMA. The NLR makes use of a method which parallelizes over the groups at fine levels, while it parallelizes over the wave directions at coarser levels. A consequence of this is that a lot of data transfer is needed during the switch between the two different ways of parallelization.
It is worth trying to parallelize the whole algorithm over the wave directions, but at the finest levels also over the groups. This possibly results in a fast way of calculating the algorithm, but there need to be enough threads to calculate all of this in parallel.

In Chapter 5, it was told the NLR uses a GMRES solver for the MLFMA. It could happen there is another solving method that is more efficient for parallel calculations, but this will not be the main focus in this thesis.

Efficient use of memory is very important for computations on a GPU. Try to let the threads communicate as less as possible and let them calculate their own little part of the algorithm.

The first focus will be on the matrix vector multiplications; the NLR knows from practical use that this part uses most of the time in the calculations.

7.2 Survey existing literature MLFMA and GPU's

There are lots of papers about the application of the Method of Moments on GPU's, about the Fast Multipole Method on GPU's for modelling particles and about the MLFMA with non-oscillatory kernels, but very little about the Multi Level Fast Multipole Method with an oscillatory kernel e^{ikr}/r.

Multi Level Fast Multipole Methods for kernels with non-oscillatory kernels have different computational complexity from those for oscillatory kernels. The most important difference is that the computational load decreases fast at coarser levels, whereas for oscillatory kernels the computational load remains the same (less groups but more wave directions). As the work is concentrated on the finest level, the MLFMA for non-oscillatory kernels is computationally comparable with the FMA for oscillatory kernels. As the focus of the report is on the Multi Level Fast Multipole Algorithm, literature for non-oscillatory kernels is skipped.

The papers about the MoM and the FMM can still be used; they can contain very useful information about the parallelization of the composition of matrices $\mathcal{L}$, V, T and A. If the matrix vector multiplication can not be accelerated that much, the composition of matrices can possibly be used to accelerate the algorithm on the GPU.
The fact that there are not many papers about could mean that parallelizing the MLFMA on a GPU does not result in a large speedup.

It is difficult to give the exact speedup achieved in different papers, because different GPU's can not be compared because of their different characteristics. Nonetheless, it will be tried to give an idea of the order of the acceleration when implementation strategies from papers are used and the speedup is given in the paper.

In the table below, different manners to accelerate the FMM or the MLFMA on GPU's are given. It can also be seen in which papers the different manners are discussed, so it is clear if it is a frequently used manner or a new or not very useful manner.
Afterwards, it will be discussed if the different manners are useful for this research and for which components they can be used.

How?	Reference
Each thread computes one entry of the near-field interaction matrix	[13],[15],[18],[32],[33]
BiCGSTAB as solver	[13],[14]
Each thread is related to one basis function	[13]
Distribute data among nodes using a group-based partitioning scheme	[13],[15],[18]
New parallel algorithm	[16]
Use of pinned memory	[18]
Different or 'own' GPU clusters	[20]
Application of HPC clusters	[25]
'Compute on-the-fly' strategy	[26]

7.2.1 Construction $\mathcal{L}'$

Following the reasoning presented by J. Guan, S. Yan and J.M. Jin in [13], for the near interaction matrix many threads are busy to carry out the assembly of a portion of the matrix. The matrixfill can be done in parallel by letting one thread compute one entry of the matrix.

Following the reasoning presented by Q. Nguyen, V. Dang, O. Kilie and E. El-Araby in [15], the same happens, but in this case the threads are grouped into blocks that are responsible for computing the elements of one row of $\mathcal{L}'$.

In [18], J. Guan, S. Yan and J.M. Jin obtain a speedup of 124.5 when constructing the near field system matrix $\mathcal{L}'$ on a GPU instead of a CPU. Each thread calculates a non-zero element of the near field system matrix and stores this in the global memory.

7.2.2 Construction V, T, A

Following the reasoning presented by Q. Nguyen, V. Dang, O. Kilie and E. El-Araby in [15], the construction of the translation matrix T works the same as the construction of $\mathcal{L}'$; each thread calculates one entry of the matrix and the threads are grouped into blocks, that are responsible for computing the elements of one row of T.
The outgoing and incoming waves V and $\overline{V}^T$ are complex conjugates of each other, so just V needs to be constructed. The group-based partitioning scheme is used for the construction, which should be done K times.

7.2.3 Solve

Following the reasoning presented in [13] and [14], it is important to have a preconditioner that can be parallelized efficiently on a GPU. The iterative method BiCGSTAB can be used together with preconditioners ILU0, Approximate Inverse (AI) or Jacobi.
From the examples in [13], it can be seen that the ILU0 is not suitable for working with GPU's because of the forward and back substitutions. The AI preconditioner is the cheapest preconditioner in the solution phase and the Jacobi is the cheapest preconditioner in the construction phase. It depends on the size of the example in the paper if the AI or the Jacobi preconditioner achieves more speedup; larger problems achieve more speedup using the AI preconditioner.

7.2.4 Matrix vector multiplication

According to the authors of [13], it seems fast evaluation of the scattered fields can be achieved by letting each thread be related to one basis function. At a specific wave direction, each GPU thread calculates the corresponding scattered field in parallel. One CPU thread superposes all the calculated scattered fields in series to avoid write conflicts on the GPU.
Unfortunately, this does not work for the MLFMA because this algorithm works with different group sizes, so different wave directions on the different levels.

Following the reasoning presented in [13],[15] and [18], the distribution of data among nodes using a group-based partitioning scheme works well on GPU's. Unfortunately, the group sizes change in the MLFMA, so this way of parallel computing does not work for the MLFMA.

The parallelization of the far field interactions can be implemented using the pinned or the global memory strategy, stated by the authors in [18]. The global memory strategy stores the outgoing and incoming waves at all levels on a single GPU. Therefore the computational efficiency is very high, because no data transfer between the host and device is needed. Unfortunately, the global memory is not very big, so this will limit the size of the problem that can be solved.
The pinned memory strategy stores the results to the pinned memory on multiple GPU's after calculating the outgoing and incoming waves. Larger problems can be solved because the pinned memory is much larger than the global memory, but data communications between host and device become unavoidable.
The global memory strategy obtains more speedup, but can not be used for very large problems. It is dependent on the size of the problem which memory strategy is most suitable.

The parallelization strategy of [18] is dividing different wave directions over different threads and letting one or several block(s) compute one group. Assuming all wave directions can be calculated by different threads on one GPU at the coarsest level, the parallelization at finer levels also fit on a single GPU. The size of a thread block is determined by the number of wave directions on the finest level. A number of blocks together represent a parent cube. The number of blocks representing a parent cube grows for coarser levels.
This method only reaches a speedup of 2.9 on a GPU instead of a CPU.

7.2.5 General

According to the authors of [16], a new parallelizable algorithm generates the FMM data structure in $O(N)$ time with complexity $O(N)$. Unfortunately, the algorithm is for the Fast Multipole Method for particles, which works different than an oscillatory kernel.

Following the reasoning presented in [20], different GPU clusters are very important for the future to solve problems even faster. Northrop Grumman invested in the creation of its own GPU cluster. This seems to be an advertising story and is not useful for this thesis.

In [25], a High Performance Computing (HPC) cluster is used to solve large scale scattering problems. The HPC clusters can overcome the memory limitation of GPU's and take advantage of the conventional CPU based cluster to achieve more acceleration.

For the MLFMA, more and more data communications between CPU and GPU are required if the problem size increases further, which results in a reduction of the computational efficiency. Therefore, [26] uses a 'compute on-the-fly' strategy, where the whole system matrix is not

computed and stored before the iterative solution. The entries are computed whenever they are needed in the matrix vector products. The number of recalculations depends on the number of iteration steps. In this case, the second-kind Fredholm integral equation is used because of its optimal iterative convergence.

7.3 Possible methods and bottlenecks

After the first ideas and reading literature about the application of the FMM and the MLFMA on GPU's, only some options are available for possibly speedup the MLFMA on GPU's. The different options from literature and some new ideas are explained below.

The previous section was divided in three different parts of the MLFMA that can possibly be accelerated on a GPU; the construction of $\mathcal{L}'$, the construction of V, T and I and the matrix vector product. In some papers, the construction of matrices is parallelized by letting each thread calculate one matrix entry. Sometimes those threads are grouped into blocks that are responsible for computing the elements of one row of $\mathcal{L}'$.
The construction of V, T and I can be done in the same way, but in those papers V and T contain information about the groups for just one wave direction, so this matrix fill should be done K times. It could be a good idea to parallelize this over the wave directions and the groups at the same time. This means that all non-zero matrix elements of V and T from our report are computed in parallel. This was not discussed in those papers but is worth trying.

In the section with some first ideas for parallelizing the MLFMA, it was told another solver can possibly perform better on a GPU than the GMRES solver. In [13] the BiCGSTAB solver is used with a good preconditioner, which is useful to try for this research as well. The Induced Dimension Reduction (IDR) method is another method which can possibly solve the problem faster on a GPU.

Parallelizing over the wave directions and using the pinned memory can speedup the algorithm. For this strategy, there need to be enough threads to be able to parallelize over the wave directions on the coarsest level.

Another strategy from literature is to recalculate matrices when they are needed. This results in less data transfer, but is also inefficient because of the large number of recalculations for large problems.

The memory use is still a difficult choice when working with GPU's, the memory is limited and there are a lot of different types. Shared memory is the type of memory which is used the most on GPU's, but the size of this memory is limited. To prevent a lack of memory, the CPU memory can be used as well.

There is nothing discussed about local or global interpolation in the papers used for this thesis. This could be something to find out in the research part of this thesis.
The same applies to the idea of parallelizing over both the wave directions and the groups. This is something to try on the GPU and see if this results in a speedup.

Chapter 8

Conclusions and Recommendations

A GPU implementation of the Multi Level Fast Multipole Method can speedup electromagnetic scattering problems. There are not many specific papers about the application of the MLFMA on GPU's, which could mean it is a difficult method for parallel computing on a GPU and does not result in a large speedup. This is something which can be answered after the research part of this thesis.

For the construction of the matrices, different papers let each thread calculate one entry of the matrix. Using this parallelization implementation, a large speedup is obtained compared to the calculations on CPU's. The matrix vector multiplications is the part which is most time consuming, so for this research the focus is on this part.

Comparing the BiCGSTAB method and the GMRES method would be a good idea. In different papers the BiCGSTAB method is used while the NLR uses the GMRES method. Comparing the GMRES method with the IDR method is can be done as well. Something else that could accelerate the MLFMA on a GPU is using pinned memory. This makes data transfer faster. If the data transfer is still too time consuming, matrices can be computed everytime they are needed, so they won't be precomputed and therefore no data transfer is needed. Global interpolation methods and local interpolation methods can also be compared. The NLR uses a local interpolation method, but it could happen a global method works faster on a GPU.

Another way of parallelizing the MLFMA on a GPU which is not discussed in the papers is parallelizing over both the wave directions and the groups. There should be enough threads to calculate all this in parallel, so this is something to find out.

To get used with the MLFMA and working on a GPU, it is advisable to start working with the before mentioned ways of accelerating the MLFMA on a GPU. Afterwards this can be accelerated even further by optimizing the discussed methods or coming up with new ideas.

Bibliography

[1] H. van der Ven and H. Schippers, 'Mathematical definition document for the multi-level fast-multipole method', NLR-TR-2010-529, 2015.

[2] S.A. Hack, 'Spherical harmonics based aggregation in the multilevel fast multipole algorithm', 2015.

[3] Ö. Ergül and L. Gürel, 'The Multilevel Fast Multipole Algorithm (MLFMA) for Solving Large-Scale Computational Electromagnetics Problems', 2014.

[4] A. Heldring, 'Full-Wave Analysis of Electrically Large Reflector Antennas', 2002.

[5] J. Jin, C. Lu and W. Chew, 'On the formulation of hybrid finite-element and boundary-integral method for 3D scattering', IEEE transactions on antennas and propagation, 1998.

[6] W. Gibson, 'The Method of Moments in Electromagnetics', 2008.

[7] K. Sertel, 'Multilevel Fast Multipole Method for Modeling Permeable Structures using Conformal Finite Elements', 2003.

[8] J.M. Song and W.C. Chew, 'Multilevel Fast Multipole Algorithm for Solving Combined Field Integral Equation of Electromagnetic Scattering', 1995.

[9] L. Zhanhea, H. Peilina, G. Xub, L. Yinga and J. Jinzua, 'Multi-frequency RCS Reduction Characteristics of Shape Stealth with MLFMA with Improved MMN', Chinese Journal of Aeronautics, vol. 23, no. 3, (2010).

[10] H. van der Ven and H. Schippers, 'High Range Resolution Profiles for a Civilian Aircraft Inlet', NLR-TR-2010-527, 2011.

[11] L. Li, J. He, Z. Liu and L. Carin, 'MLFMA Analysis of Scattering from Multiple Targets In the Presence of a Half Space', 2003.

[12] E. Darve, 'The Fast Multipole Method: Numerical Implementation', 1999.

[13] 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, (2014).

[14] Ö. Tayfun, J. Malcolm and M. Yuriy, 'Multi-Core CPU and GPU Accelerated FMM-FFT Solver for Antenna Co-Site Interference Analysis on Large Platforms', AP-S, 2014.

[15] Q. Nguyen, V. Dang, O. Kilie and E. El-Araby, 'Parallelizing Fast Multipole Method for Large-Scale Electromagnetic Problems Using GPU Clusters', IEEE antennas and wireless propagation letters, vol. 12, (2013).

[16] Q. Hu, N. Gumerov and R. Duraiswami, 'Scalable Fast Multipole Methods on Distributed Heterogeneous Architectures', University of Maryland.

[17] A. Chandramowlishwaran, S. Williams, L. Oliker, I. Lashuk, G. Biros and R. Vudue, 'Optimizing and Tuning the Fast Multipole Method for State-of-the-Art Multicore Architectures', IEEE, 2010.

[18] J. Guan, S. Yan and J. 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, (2013).

[19] 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, (1993).

[20] K. D'Ambrosio and R. Pirich 'Parallel Computation Methods for Enhanced MOM and MLFMM Performance', IEEE, 2009.

[21] M. Cwikla 'Low-Frequency MLFMA on Graphics Processors', IEEE, 2010.

[22] D.M. Hailu 'Hybrid Spectral-Domain Ray Tracing Method for Fast Analysis of Millimeter-Wave and Terahertz-Integrated Antennas', IEEE, 2011.

[23] B. Kolundzija, M. Tasic, D. Olcan, D. Zoric and S. Stevanetic 'Full-Wave Analysis of Electrically Large Structures on Desktop PCs', IEEE, 2011.

[24] E. Yunis, R. Yokota and A. Ahmadia 'Scalable Force Directed Graph Layout Algorithms Using Fast Multipole Methods', IEEE, 2012.

[25] V. Dang, Q. Nguyen, O. Kilic and E. El-Araby 'Single Level Fast Multipole Method on GPU cluster for Electromagnetic Problems', IEEE, 2013.

[26] J. Guan, S. Yan and JM. Jin 'A GPU-Accelerated Integral-Equation Solution for Large-Scale Electromagnetic Problems', IEEE, 2014.

[27] D. Faircloth, T. Killian, M. Horn, M. Shafieipour, I. Jeffrey, J. Aronsson and V. Okhmatovski 'Fast Direct Higher-Order Solution of Complex Large Scale Electromagnetic Scattering Problems via Locally Corrected Nystrom Discretization of CFIE', IEEE, 2014.

[28] N. Tran, T. Phan and O. Kilic 'Analysis of Micro-Doppler Signature Due To Indoor Human Motion Using Multilevel Fast Multipole Algorithm On GPU Cluster', IEEE, 2015.

[29] J. Guan 'OpenMP-CUDA Implementation of the Moment Method and Multilevel Fast Multipole Algorithm on Multi-GPU Computing Systems', Thesis at University of Illinois, 2013.

[30] W. C. Chew, J-M. Jin, E. Michielssen and J. Song 'Fast and Efficient Algorithms in Computational Electromagnetics', Artech House, 2001.

[31] N. Gupta, `http://cuda-programming.blogspot.nl/2013/01/thread-and-block-heuristics-in-cuda.html`.

[32] M. Lopez-Portugues, J. A. Lopez-Fernandez, J. Menedez-Canel, A. Rodriguez-Campa and J. Ranilla 'Acoustic scattering solver based on single level FMM for multi-GPU systems', Elsevier, 2011.

[33] M. Lopez-Portugues, J. A. Lopez-Fernandez, A. Rodriguez-Campa and J. Ranilla 'A GPGPU solution of the FMM near interactions for acoustic scattering problems', Springer, 2011.

[34] R. Erkamp, `http://www.cs.vu.nl/~rob/masters-theses/Ronald-Erkamp.pdf`.

Appendix A

List of symbols

Symbol	Explanation
A_l^{l-1}	Agglomeration matrix.
b	Incoming radar wave.
c_m	Group center of degree of freedom r.
$c_{m'}$	Group center of degree of freedom r'.
d_m	Distance between r and c_m.
D	Distance between r' and c_m.
f_n	Basis function.
g_k	Lagrange interpolated function.
G_l	Number of groups on level l.
$h_l^{(1)}$	Spherical Hankel function of the first kind.
I_l^{l-1}	Lagrange interpolation matrix.
j_l	Spherical Bessel function of the first kind.
J	Vector that needs to be determined.
k	Wave number.
$\hat{k}$	Wave direction.
K_l	Number of integration points used in the k-space integral at level l.
$\mathcal{L}'$	Near field interactions between different scatterers.
M	Number of basis functions in one group.
N	Total number of basis functions.
P_l	Legendre polynomial.
r, r'	Degree of freedom.
$r_{mm'}$	Distance between c_m and $c_{m'}$.
S	Surface of a bounded scatterer.
T	Translation matrix.
V	Incoming/outgoing wave information.
w	Weight.
x	Unknown currents on the airplane.
θ, ϕ	Spherical coordinates.

Appendix B

Survey existing literature MLFMA and GPU's

B.1 Paper 1: An Accurate and Efficient Finite Element-Boundary Integral Method With GPU Acceleration for 3-D Electromagnetic Analysis

Following the reasoning presented by [13], the following notes kan be made.

- The accuracy of the FE-BI method is improved using Rao-Wilton-Glisson or Buffa-Christiansen functions as testing functions.
- To accelerate the convergence of the iterative solution in the FE-bi(CFIE) method, the absorbing boundary condition (ABC)-based preconditioner is used.
- To further improve the efficiency of the total computation, the MLFMA is applied to the iterative solution.
- Using multi-GPU computing systems, the assebly of the near field matrices could be calculated on multiple GPU's, the calculations of the inner and outer iterations could be accelerated and the scattered fields could be evaluated.
- The radiation patterns and receiving patterns on each level can be calculated in parallel.
- To obtain a maximum parallel efficiency on all levels, the implementation strategy 'one thread per spectrum sampling' and 'one/several block(s) per group' is adopted, so that groups and their far field patterns partitioned simultaneously.
- An inner iteration scheme will be developed, which can be parallelized efficiently on GPU's.
- BiCGSTAB with the ILU0, the AI and the Jacobi preconditioners are employed to solve $([A]) + [M])\{y\} = \{u\}$, where y the unknown.
    1. The ILU0 preconditioner requires roughly the same amount of memory as the input matrix $([A])+[M])$, but forward and back substitutions make it inefficient for parallel processing on GPU's.

2. The AI preconditioner is an incomplete approximation of the inverse of the input matrix based on the minimization of the Frobenius norm. In contrast to the ILU-based preconditioners, the AI preconditioner is applied using MVPS, which makes it very suitable for GPU parallelization.

3. The Jacobi preconditioner is very cheap to generate and apply, but the iterative convergence is slow when the input matrix is ill-conditioned.

- Since the basis functions can be regarded as the current sources from which the scattered fields are generated, a one dimensional grid of threads is allocated on the GPU, and each thread is related to one basis function.
- For the scattered field at a specific observation angle, each GPU thread calculates the corresponding scattered field in parallel, and one CPU thread superposes all the calculated scattered fields in series to avoid write conflicts on the GPU.
- To further accelerate the scattered field evaluation, the OpenMP parallel technique is employed to generate multiple CPU threads, and each CPU thread manages one GPU device to calculate a portion of the scattered fields.
- Two methods for solving $([A]) + [M])\{y\} = \{u\}$:
  1. Direct method solve: $([A]) + [M])$ is first decomposed into a lower and an upper triangular matrix, and solved by the forward and back substitutions.
  2. Iterative method solve: (ILU0, AI or Jacobi) preconditioned BiCGSTAB method with a targeted relative residual error of 10^{-3} is applied for the solve.
- The direct method is expensive in the construction phase and cheap in the solution phase. However, the direct method will require a larger storage and a higher computational cost when the problems become large, and the solution is very difficult to parallelize on GPU's. Therefore only the interative methods will be considered for the GPU calculation.
- Conclusions of the three preconditioners:
  - AI preconditioned iterative method is most efficient in the solution phase.
  - Jacobi is cheapest in the construction phase.
  - ILU0 is not well suited for the GPU parallelization, because it requires forward and bck substitutions.
  - More speedup using AI than Jacobi in the human body and roar object examples, but it also needs more memory.
- Modified ABC-based preconditioner is better than the original preconditioner, because it is symmetric and more effective.
- For further speedup the computation, a GPU-accelerated FE-BI algorithm was developed.
- The FE-BI(CFIE) method is accurate, robust and efficient for practical applications.

B.2 Paper 2: Multi-Core CPU and GPU Accelerated FMM-FFT Solver for Antenna Co-Site Interference Analysis on Large Platforms

Following the reasoning presented by [14], the following notes kan be made. Important information: This paper used a FMM-FFT solver, which is different from ours.

- The single-level toeplitz grouping strategy is much easier to parallelize and results in much better scaling than the multi-level FMM approach.
- The BiCGstab method can be accelerated on a multi-core CPU. The matrix fill can be accelerated the most: up to 18 times. The solver can be accelerated up to 14 times.
- The BiCGstab method can also be accelerated on a GPU wrt a single thread CPU. The acceleration of the matrix fill is up to 116,8 times and the solver 7 times.

B.3 Paper 3: Parallelizing Fast Multipole Method for Large-Scale Electromagnetic Problems Using GPU Clusters

Following the reasoning presented by [15], the following notes kan be made.

- Uniformly distributing the total number of groups M among the N computing nodes. Each node holds the same amount of workload.
- Rows of Z_{near} are assigned to the computing nodes following the group-based distirbution. Each node has approximately an equal number of near interactions, and they are calculated without any communication with the other nodes.
- At a given node, the computation of each element Z_{mn} is performed by one thread enabling all row computations to be allocated to one CUDA block.
  - The Z_{near} matri is partitioned among the computing nodes, such that each CUDA thread per node is assigned to compute one element Z_{mn}, and threads are grouped into blocks that are responsible for computing the elements of one row of Z_{near} for that node.
  - This results in a total number of CUDA blocks (i.e. rows in the matrix) per node equal to $N_{group}M_{node}$.
- Radiation/receive function calculations: Data is distributed among nodes and the calculations per node are performed such that M_{node} groups are allocated for each node. Each group per node requires K evaluations of the radiation/receiving function. The threads are grouped into blocks such that each block of threads performs N_{group} radiation calculations at a given direction, resulting in a number of blocks per node equal to $M_{node}K$.
- The far field translation is the same as near interactions: each CUDA block is assigned to compute one row of the T_l matrix at a given direction, which results in a total number of thread blocks per node equal to $M_{node}K$.
- For the fast matrix-vector multiplication is the group-based distribution scheme used: the internode communication is required only at two steps:

1. At the beginning of the matrix-vector multiplication to exchange the estimated values for the unknowns among the nodes.
2. After the aggregation step and before performing the translation step in order to update all nodes with the current aggregation results.

- Before entering the linear system solution, each node calculates its own portion of the radiation/receive functions and the translation matrix.
- During the iterative solution, each node also calculates the estimates values of the unknowns. It is important to note that the distribution of the unknowns among the computing nodes follows the same group-based distribution of the radiate/receive functions.
- In the aggregation step, each unknown is multiplied by its corresponding radiation function and is accumulated within a given group in order to calculate the total field for that group.
- After aggregation, an all-to-all communication is employed by each node to broadcast the aggregated fields to all other nodes.
- For larger problems, it should not be forgotten that the GPU memory has a limit. A solution for this problem is to utilize both CPU and GPU memory spaces.

B.4 Paper 4: Scalable Fast Multipole Methods on Distributed Heterogeneous Architectures

Following the reasoning presented by [16], the following notes kan be made.

- Steps for parallelization on GPU's:
  - Determine of the Morton index for each particle.
  - Construction of occupancy histogram and bin sorting.
  - Parallel scan and global particle ranking.
  - FInal filtering.
  - Final bin sorting.
- Determining the interacting source boxes in the nieghborhood of the receiver boxes.
- Different stages of the FMM have very different efficiency when parallelized on the GPU:
  - Lowest efficiency (due to limited GPU local memory) is for translations.
  - Computation of $A^{near}J$ on GPU is very efficient.
  - Anything having to do with particles is very efficient on the GPU, and translations are relatively efficient on CPU.

B.5 Paper 5: Optimizing and Tuning the Fast Multipole Method for State-of-the-Art multicore Architectures

Following the reasoning presented by [17], the following notes kan be made. In this paper a lot of different GPU's are discussed. This is not very relevant for this thesis.

- Taking square roots and divide operations are very slow on GPU's.
- Mistakes
  - Relied on bulk-synchronous parallelism; some working sets did not fit in cache. Dataflow and work-queue approaches may mitigate this issue.
  - Optimal algorithmic parameters will vary not only with architecture, but also optimization and scale of parallelism.
  - Manual SIMD transformations and their interaction with data layout was a significant performance win, and should be a priority for new compile and/or programming model efforts.

B.6 Paper 6: An OpenMP-CUDA Implementation of Multilevel Fast Multipole Algorithm for Electromagnetic Simulation on Multi-GPU Computing Systems

Following the reasoning presented by [18], the following notes kan be made.

- On finer levels, groups at the same level are partitioned into different processors and each processor gets approximately the same number of groups.
- On coarser levels, far-field patterns (FFPs) are replicated for every processor. When the number of processors increases, this parallelization strategy is not effective around the transition levels where neither the number of groups, nor the number of FFPs is large enough to achieve a good parallel efficiency.
- Examples of 'one thread per observer' are: parent group in the aggregation phase, child-group in the disaggregation phase and destination group in translation phase. This is getting a low parallel efficiency when the number of groups decreases at coarser levels.
- Curvilinear Rao-Wilton-Glisson (CRWG) functions are used as the basis functions.
- The intensive parts are:
  - Iterative solution
  - Radiation patterns of the basis functions V_s
  - Receiving patterns of the testing functions V_f
  - Translator T
  - Assembly of near-field system matrix Z_{near}
- For the far field interaction: parallelly computing the radiation patterns and receiving patterns of the groups in the aggregation, translation and disaggregation phases.

- In the aggregation phase, the size of thread block in the algorithm is set as the size of the spectrum at the finest level for the optimal use of hardware resources.
- Global memory strategy: radiation and receiving paytterns should be calculated and stored at all levels on a single GPU. This avoids data transfer between host and device and is therefore fast! But the size of the global memory will limit the size of the problems that can be solved.
- Pinned memory strategy: calculates radiation and receiving patterns on multiple GPU's and stores the results in the pinned memory on the host. This could solve larger problems, because the size of the pinned memory is much larger. But data communications between host and device become unavoidable.