A PARALLEL SHARED MEMORY IMPLEMENTATION OF THE FAST MULTIPOLE METHOD FOR ELECTROMAGNETICS

MARTIN NILSSON[2] *

[2]*Department of Information Technology, Scientific Computing*
Uppsala University, SE-75105 Uppsala, Sweden. email: martin@tdb.uu.se

Abstract

An implementation of the multilevel Fast Multipole method for time harmonic electromagnetic computations is presented. The method is parallelized for shared memory machines. A new parallelization scheme which is a hybrid between previous methods is proposed. Several symmetries and other methods that reduce the storage requirement are proposed. The most important symmetry is for the translation operators. A method for reducing the CPU-time in the interpolation is also proposed.

Keywords: parallel, shared memory, Fast Multipole method, implementation, electromagnetics

AMS subject classification (MSC2000): 65R20, 65Y05, 78M15

1 Introduction

The Method of Moments is a popular method in the computational electromagnetics community. It is a boundary integral equation method for solving the time-harmonic Maxwell's equations. It can be used to model the electromagnetic properties of both small and large objects. A drawback of the method is that the matrix for the linear equation system is dense. The resulting storage requirement is then $\mathcal{O}(N^2)$, where N is the number of unknowns. This effectively prohibits the solution of problems larger than the order of 10^5 unknowns unless the largest supercomputers are used. The solution time is also limiting

*Financial support has been obtained from Parallel and Scientific Computing Institute (PSCI), which is a competence center financed by Vinnova, The Swedish Agency for Innovation Systems, and the Swedish National Aeronautical Research Program, NFFP.

since Gaussian elimination requires $\mathcal{O}(N^3)$ arithmetic operations and iterative methods, e.g. GMRES, QMR, require $\mathcal{O}(MN^2)$ arithmetic operations where M is the number of iterations. Since N depends squarely on the electrical size of the object, electrically large problems can not be solved. The Fast Multipole method was proposed as a remedy to the problem by Rokhlin in [13] and later by Coifman et. al in [4]. The idea is to diagonalize the Green's function analytically to compute a matrix vector multiplication faster than $\mathcal{O}(N^2)$. The Fast Multipole method was later extended in [14] to a multilevel version which has been used extensively in for instance [3, 5, 9] and references therein. The multilevel Fast Multipole method requires $\mathcal{O}(N \log N)$ memory and $\mathcal{O}(MN \log N)$ arithmetic operations.

In this paper we present an implementation of the multilevel Fast Multipole method. Although the asymptotic scaling of the multilevel Fast Multipole method is $\mathcal{O}(N \log N)$ the constant multiplying the leading term is quite large. Therefore it is important to reduce the constant as much as possible. The contributions of this paper are symmetries that reduce the storage requirement for the different translation operators, simple methods for reducing the memory requirement and a reduction of the CPU-time for interpolation. Previous efforts to reduce the memory consumption and CPU-time can be found in [3, 5, 6]. The implementation is parallelized for shared memory multiprocessor machines. A block parallel hybrid method proposed in this paper is compared to a simple parallel implementation and the method in [15]. The previous two methods can be summarized as parallelization over boxes and parallelization over quadrature points. The hybrid method combines the two methods by blocking over boxes, yielding greater opportunity for parallelization over quadrature points. In numerical experiments we show that the new method improves the parallel scalability. We also examine the effect of some of the memory reductions proposed. Although we only show results on perfect electric conductors, the methods can also be used on other materials like homogenous dielectrics.

The rest of this paper is organized as follows: In Section 2 the integral equations for a perfect electric conductor are described. Section 3 describes the approximations that lead to the Fast Multipole method. Section 4 and Section 5 describes the Fast Multipole algorithm. We describe different implementation issues and our contributions in Section 6. The parallelization strategies are presented in Section 7. Experimental results are presented in Section 8 and conclusions are given in Section 9.

2 Integral equations

Consider the time-harmonic electromagnetic scattering from a perfect electric conductor (PEC). Combining the Electric Field Integral Equation (EFIE) and the Magnetic Field Integral Equation (MFIE) in variational form yields the Combined

Field Integral Equation (CFIE) [11]

$$\begin{aligned}
&\alpha \int_\Gamma \int_\Gamma G(\mathbf{x}, \mathbf{x}') \left(\mathbf{J} \cdot \mathbf{J}' - \frac{1}{\kappa^2} \boldsymbol{\nabla}_\Gamma \cdot \mathbf{J} \boldsymbol{\nabla}_\Gamma \cdot \mathbf{J}' \right) d\Gamma d\Gamma \\
&+ (1-\alpha) \frac{\imath}{\kappa} \int_\Gamma \hat{\mathbf{n}} \times \int_\Gamma \boldsymbol{\nabla}_{\mathbf{x}'} G(\mathbf{x}, \mathbf{x}') \times \mathbf{J} \cdot \mathbf{J}' d\Gamma d\Gamma \\
&= -\alpha \frac{1}{\imath \kappa Z} \int_\Gamma \mathbf{E}_a \cdot \mathbf{J}' d\Gamma + (1-\alpha) \frac{\imath}{\kappa} \int_\Gamma \hat{\mathbf{n}} \times \mathbf{H}_a \cdot \mathbf{J}' d\Gamma.
\end{aligned} \tag{1}$$

Here, $\mathbf{J}$ is the unknown electric current on the surface Γ of the scatter, $\mathbf{J}'$ is the test current, κ is the wavenumber, Z is the impedance in free space, $\hat{\mathbf{n}}$ is the unit normal pointing outward from Γ, and $\imath = \sqrt{-1}$. The function $G(\mathbf{x}, \mathbf{x}')$ is the free-space Green's function for Helmholtz' equation. The parameter α can vary between 0 (MFIE) and 1 (EFIE). The right hand side depends on the applied electric field $\mathbf{E}_a$ and the applied magnetic field $\mathbf{H}_a$.

The equations are discretized with the Galerkin method and the rooftop or RWG basis functions [12] with $\mathbf{J}(\mathbf{x}) = \sum_{k=1}^{N} I_k \mathbf{j}_k(\mathbf{x})$. The discretization leads to a dense, complex system of equations of the form $\mathbf{ZI} = \mathbf{V}$ [11, 12]. The unknowns in $\mathbf{I}$ are the coefficients for each basis function and the right hand side $\mathbf{V}$ depends on the applied fields $\mathbf{E}_a$ and $\mathbf{H}_a$. A change in the applied field affects only the right hand side, while $\mathbf{Z}$ is unchanged.

3 Approximating the Green's function

The Fast Multipole Method [3, 4] uses an approximation of the Green's function for Helmholtz' equation which is based on two observations. The first observation is that the Green's function for Helmholtz' equation can be expanded into an infinite series using Gegenbauer's addition theorem [1]. It states that

$$\begin{aligned}
\frac{e^{\imath\kappa|\mathbf{X}+\mathbf{d}|}}{|\mathbf{X}+\mathbf{d}|} &= \imath\kappa h_0^{(1)}(\kappa|\mathbf{X}+\mathbf{d}|) \\
&= \imath\kappa \sum_{l=0}^{\infty} (-1)^l (2l+1) j_l(\kappa d) h_l^{(1)}(\kappa X) P_l\left(\hat{\mathbf{d}} \cdot \hat{\mathbf{X}}\right)
\end{aligned} \tag{2}$$

where $j_l(x)$ is a spherical Bessel function, $h_l^{(1)}(x)$ is a spherical Hankel function and $P_l(x)$ is a Legendre polynomial. The length of a vector $\mathbf{x}$ is given by $x = |\mathbf{x}|$ and a unit length vector is denoted with a $\hat{}$ above it. The expansion is valid as long as $d < X$. It can be used to compute the field at a receiver location $\mathbf{x}$ from a source location $\mathbf{x}'$. The choice $\mathbf{d} = \mathbf{x} - \mathbf{X}_m + \mathbf{X}'_m - \mathbf{x}'$ and $\mathbf{X} = \mathbf{X}_m - \mathbf{X}'_m$ implies that $d < X$ if $|\mathbf{x} - \mathbf{X}_m + \mathbf{X}'_m - \mathbf{x}'| < |\mathbf{X}_m - \mathbf{X}'_m|$, which is the case if $|\mathbf{x} - \mathbf{X}_m| < X/2$ and $|\mathbf{x}' - \mathbf{X}'_m| < X/2$. The second observation is that the

product $j_l(\kappa d) P_l\left(\hat{\mathbf{d}} \cdot \hat{\mathbf{X}}\right)$ can be expanded into an integral over propagating plane waves [4]

$$4\pi \imath^l j_l(\kappa d) P_l\left(\hat{\mathbf{d}} \cdot \hat{\mathbf{X}}\right) = \int_{\mathcal{S}} e^{\imath \boldsymbol{\kappa} \cdot \mathbf{d}} P_l\left(\hat{\boldsymbol{\kappa}} \cdot \hat{\mathbf{X}}\right) d^2\hat{\boldsymbol{\kappa}} \tag{3}$$

Here, the integral is taken over the unit sphere $\mathcal{S}$ and $\boldsymbol{\kappa} = \kappa\hat{\boldsymbol{\kappa}}$ is the direction of the propagating plane wave.

It turns out that only a finite number $L+1$ terms are needed in the sum (2) to obtain a desired accuracy in an approximation of the Green's function. To control the error ϵ one must decide where to truncate the infinite sum. Here we will use the formula in [3]

$$L = \kappa d + \beta (\kappa d)^{\frac{1}{3}} \tag{4}$$

where $\beta = 1.8(-\log_{10}(\epsilon))^{\frac{2}{3}}$ determines the number of digits. When the number of terms is determined, inserting equation (3) into equation (2) and changing the order of summation and integration yields

$$\frac{e^{\imath\kappa|\mathbf{X}+\mathbf{d}|}}{|\mathbf{X}+\mathbf{d}|} \approx \frac{\imath\kappa}{4\pi} \int_{\mathcal{S}} e^{\imath\boldsymbol{\kappa}\cdot\mathbf{d}} \sum_{l=0}^{L} \imath^l (2l+1) h_l^{(1)}(\kappa X) P_l\left(\hat{\boldsymbol{\kappa}} \cdot \hat{\mathbf{X}}\right) d^2\hat{\boldsymbol{\kappa}} \tag{5}$$

In order to evaluate the integral in (5) a quadrature formula that is exact for the first $2L$ spherical harmonics is needed [4]. The choice of $2L+2$ equidistant nodes in the azimuthal angle ϕ and $L+1$ Gauss-Legendre quadrature nodes in the elevation angle θ is sufficient [5]. Note that this choice implies that if (θ, ϕ) is a quadrature point, then $(\pi - \theta, \phi + \pi)$ is also a quadrature point, i.e. if plane wave direction $\hat{\boldsymbol{\kappa}}$ is a quadrature point then so is $-\hat{\boldsymbol{\kappa}}$. The choice gives a total of $K = (2L+2)(L+1)$ quadrature points. With $\mathbf{X} = \mathbf{X}_m - \mathbf{X}'_m$ and $\mathbf{d} = \mathbf{x} - \mathbf{X}_m + \mathbf{X}'_m - \mathbf{x}'$

$$\frac{e^{\imath\kappa|\mathbf{x}-\mathbf{x}'|}}{4\pi|\mathbf{x}-\mathbf{x}'|} \approx \sum_{k=1}^{K} e^{\imath\boldsymbol{\kappa}_k\cdot(\mathbf{x}-\mathbf{X}_m)} \mathcal{T}_k^L(\kappa, \mathbf{X}_m - \mathbf{X}'_m) e^{\imath\boldsymbol{\kappa}_k\cdot(\mathbf{X}'_m-\mathbf{x}')} \tag{6}$$

Here, $\mathcal{T}_k^L(\kappa, \mathbf{X})$ is the translation operator defined by

$$\mathcal{T}_k^L(\kappa, \mathbf{X}) = \frac{\imath\kappa}{16\pi^2} w_k \sum_{l=0}^{L} \imath^l (2l+1) h_l^{(1)}(\kappa X) P_l\left(\hat{\boldsymbol{\kappa}}_k \cdot \hat{\mathbf{X}}\right) \tag{7}$$

where w_k is the quadrature weight associated with the plane wave direction $\hat{\boldsymbol{\kappa}}_k$. Note that equation (6) is true for any source close to $\mathbf{X}'_m$ and any receiver close to $\mathbf{X}_m$ as long as $d < X$ and that $\mathcal{T}_k^L(\kappa, \mathbf{X}_m - \mathbf{X}'_m)$ is independent of the locations of sources and receivers.

The spherical Hankel functions $h_l^{(1)}(x)$ becomes highly oscillatory for $l \to \infty$ and x fixed. This means that the operator $\mathcal{T}_k^L(\kappa, \mathbf{X})$ becomes numerically unstable for large values of L. A solution to the problem is to use the exact expression for small arguments of $x = \kappa X$ and use the approximate expression for large arguments. Depending on the smallest size of x, the size of L is also limited. Thus, there is a lower bound for the precision in equation (5) that depends on the smallest size of the argument.

4 Fast Multipole Method for CFIE

The Green's function approximation (6) applied to a matrix element Z_{kl} in the RWG discretized CFIE (1) yields

$$Z_{kl} \approx \sum_{j=1}^{K} \mathbf{R}_k(\hat{\boldsymbol{\kappa}}_j) \cdot \mathcal{T}_j^L(\kappa, \mathbf{X}_m - \mathbf{X}'_m) \mathbf{F}_l(\hat{\boldsymbol{\kappa}}_j) \tag{8}$$

where, $\mathbf{F}_l(\hat{\boldsymbol{\kappa}})$ is called the far field pattern and $\mathbf{R}_k(\hat{\boldsymbol{\kappa}})$ the receiving pattern. They are defined by

$$\begin{cases} \mathbf{F}_l(\hat{\boldsymbol{\kappa}}) = \hat{\boldsymbol{\kappa}} \times \int_\Gamma e^{\imath\boldsymbol{\kappa}\cdot(\mathbf{X}'_m - \mathbf{x}')} \mathbf{j}_l d\Gamma(\mathbf{x}') \times \hat{\boldsymbol{\kappa}} \\ \mathbf{R}_k(\hat{\boldsymbol{\kappa}}) = \alpha\hat{\boldsymbol{\kappa}} \times \displaystyle\int_\Gamma e^{\imath\boldsymbol{\kappa}\cdot(\mathbf{x} - \mathbf{X}_m)} \mathbf{j}'_k d\Gamma(\mathbf{x}) \times \hat{\boldsymbol{\kappa}} \\ \qquad\qquad + (1-\alpha) \displaystyle\int_\Gamma e^{\imath\boldsymbol{\kappa}\cdot(\mathbf{x} - \mathbf{X}_m)} \mathbf{j}'_k \times \hat{\mathbf{n}}(\mathbf{x}) d\Gamma(\mathbf{x}) \times \hat{\boldsymbol{\kappa}} \end{cases} \tag{9}$$

Because of the support of the basis functions the integrals for $\mathbf{F}_l(\hat{\boldsymbol{\kappa}})$ and $\mathbf{R}_k(\hat{\boldsymbol{\kappa}})$ are local. An important point is that $\mathbf{F}_l(\hat{\boldsymbol{\kappa}})$ only has two components, since $\hat{\boldsymbol{\kappa}} \cdot \mathbf{F}_l(\hat{\boldsymbol{\kappa}}) = 0$. Thus, the scalar product in (8) is between two components.

Approximation (8) can be used to accelerate the calculation of a matrix vector product **ZI**. The acceleration comes from the fact that several far field patterns from basis functions close to each other can use the same translation operator $\mathcal{T}_k^L(\kappa, \mathbf{X})$ to transfer their contributions to another area with receiving patterns from other basis functions. In order to do this in a systematic way the basis function of the object are clustered into boxes.

A simple way to cluster the basis functions together is to create a tree structure with boxes that are associated with the position of the basis functions in space. This can be achieved by enclosing the object in a box. In each direction x, y and z the sides are divided by two, resulting in 8 smaller boxes. Each one of the boxes is then divided again recursively until the size of the smallest boxes is of the required size as in Figure 1. Only the boxes with basis functions inside them are kept, so the tree structure is sparse. Another way is to select a size of the smallest boxes and assign a number to each box based on its position as

in Figure 2. With a suitable numbering of the boxes, like Morton ordering [16], it is possible to calculate the father of a box with simple bit manipulations. In this way the tree is constructed from the bottom and up. This is a better way of creating the boxes, because it is very easy to generate a list of pointers to the boxes at each level. An operation that involves all the boxes at a level is then performed by looping over the list of pointers. For an application where the same tree structure is used several times this is more efficient.

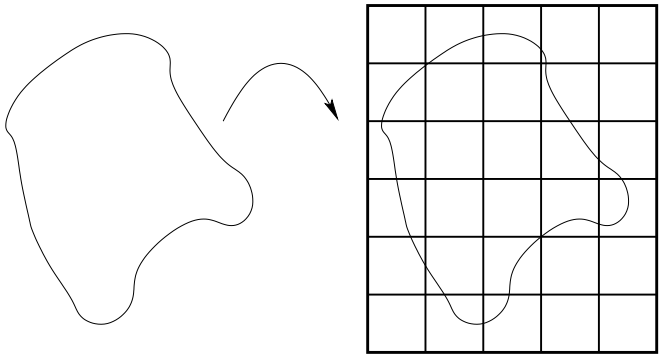

Figure 1: The object is enclosed in a box which is divided into boxes of required size.

Assume that a list with all boxes containing basis functions exists. The boxes are denoted $\mathcal{B}_m$ for $1 \leq m \leq M$, where $\mathcal{B}_m$ is box number m in the list and M is the total number of boxes. Let d be the diameter of the smallest sphere enclosing a box and X the distance between the midpoints of two boxes. When it is possible the interactions between boxes are calculated with approximation (8). When the boxes are close the criterion $d < X$ is not satisfied, which means that (8) is not valid. Thus the elements in $\mathbf{Z}$ corresponding to interaction between basis functions where $d > X$ have to be computed with a standard method.

For each box $\mathcal{B}_m$ two interaction lists are associated as in Figure 2. The first list $\mathcal{I}_m^{\mathcal{N}}$ contains the nearest neighbor boxes where $d < X$ is not true. The second list $\mathcal{I}_m^{\mathcal{F}}$ contains the boxes where $d < X$. For box $\mathcal{B}_m$, the entries in the matrix $\mathbf{Z}$ corresponding to basis functions in boxes from the list $\mathcal{I}_m^{\mathcal{N}}$ are computed and stored in memory. This part of the matrix is denoted $\mathbf{Z}^{near}$. The matrix $\mathbf{Z}^{near}$ is defined as

$$Z_{kl}^{near} = \begin{cases} Z_{kl} & k \in \mathcal{B}_m \quad \text{and} \quad l \in \mathcal{B}_n \quad \text{and} \quad \mathcal{B}_n \in \mathcal{I}_m^{\mathcal{N}} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

The algorithm that is used for approximation (8) is called the Fast Multipole Method. It needs three initialization steps. In the first step the far field pattern of each box is generated. This is achieved by computing $\mathbf{F}_l(\hat{\boldsymbol{\kappa}}_j)$ for each basis function $\mathbf{j}_l$ and each quadrature point $\hat{\boldsymbol{\kappa}}_j$ on the sphere. Since $\mathbf{F}_l(\hat{\boldsymbol{\kappa}}_j)$ has two components, a total of $2NK$ entries have to be computed. In the second step

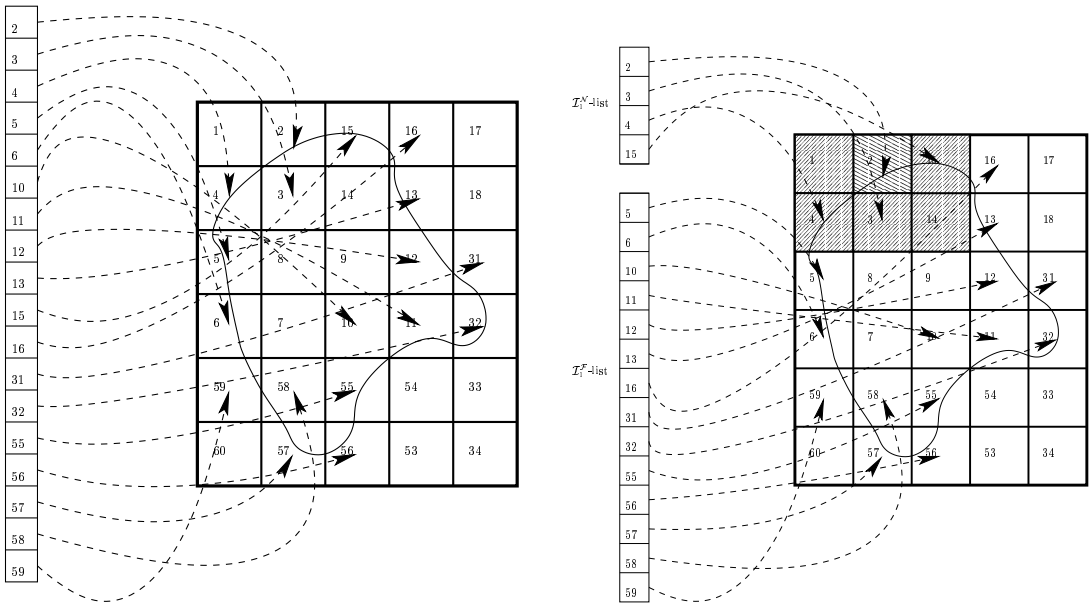

Figure 2: The boxes containing basis functions are stored in a list of pointers (left). The two interaction lists $\mathcal{I}_1^{\mathcal{N}}$ and $\mathcal{I}_1^{\mathcal{F}}$ for box $\mathcal{B}_1$. According to the numbering of the boxes in the figure, box 2 is the first box that contains basis functions (right).

all the translation operators $\mathcal{T}_j^L(\kappa, \mathbf{X}_m - \mathbf{X}_n)$ needed by a box are computed and stored. Here, $\mathbf{X}_m$ denotes the geometrical center of box $\mathcal{B}_m$ and $\mathbf{X}_n$ the geometrical center of $\mathcal{B}_n$. Each box $\mathcal{B}_m$ has to know the translation operators for all boxes in the list $\mathcal{I}_m^{\mathcal{F}}$. The number of entries that have to be computed for box $\mathcal{B}_m$ is $N_{\mathcal{I}_m^{\mathcal{F}}} K$, where $N_{\mathcal{I}_m^{\mathcal{F}}}$ is the number of boxes in $\mathcal{I}_m^{\mathcal{F}}$. The third step is to compute the receiving patterns $\mathbf{R}_k(\hat{\boldsymbol{\kappa}}_j)$ for each basis function. This also requires a total of $2NK$ entries as for the far field pattern.

With the far field patterns of box $\mathcal{B}_m$ defined by

$$\tilde{\mathbf{F}}_m(\hat{\boldsymbol{\kappa}}_j) = \sum_{l \in \mathcal{B}_m} \mathbf{F}_l(\hat{\boldsymbol{\kappa}}_j) I_l \tag{11}$$

and the translations to the receiving box $\mathcal{B}_m$ from its far field boxes defined by

$$\mathbf{G}_m(\hat{\boldsymbol{\kappa}}_j) = \sum_{\mathcal{B}_n \in \mathcal{I}_m^{\mathcal{F}}} \mathcal{T}_j^L(\kappa, \mathbf{X}_m - \mathbf{X}_n) \tilde{\mathbf{F}}_n(\hat{\boldsymbol{\kappa}}_j) \tag{12}$$

it is possible to compute $\sum_{l=1}^{N} Z_{kl} I_l$, where basis function $k \in \mathcal{B}_m$, in an efficient manner from

$$\sum_{l=1}^{N} Z_{kl} I_l \approx \sum_{l \in \mathcal{B}_j \in \mathcal{I}_m^{\mathcal{N}}} Z_{kl}^{near} I_l + \sum_{j=1}^{K} \mathbf{R}_k(\hat{\boldsymbol{\kappa}}_j) \cdot \mathbf{G}_m(\hat{\boldsymbol{\kappa}}_j) \tag{13}$$

The translation $\mathbf{G}_m(\hat{\boldsymbol{\kappa}}_j)$ is the same for all basis functions in box $\mathcal{B}_m$ and the far field pattern $\tilde{\mathbf{F}}_m(\hat{\boldsymbol{\kappa}}_j)$ is independent of the receiving box. Thus, they are calculated only once. The resulting algorithm is the single-stage Fast Multipole Method. Some analysis of the algorithm [4, 13] shows that the computation of the matrix vector requires $aNM + bN^2M^{-1}$ arithmetic operations. The total operation count is minimized if $M = \sqrt{bNa^{-1}}$, where a and b are machine and implementation dependent parameters. The result is an $\mathcal{O}\left(N^{\frac{3}{2}}\right)$ algorithm for computing the matrix vector product.

5 The multilevel Fast Multipole Method

The strategy that allows a fast matrix vector multiplication in the Fast Multipole Method can be extended to an $\mathcal{O}(N \log N)$ method [3, 5, 14]. Let us consider the multilevel tree structure. The levels are numbered $1 \leq \lambda \leq \lambda_{max}$, where 1 is the level with the smallest boxes and λ_{max} is the level with the largest boxes. For a box $\mathcal{B}_{m\lambda}$ at a given level λ, matrix elements from the nearest neighbor boxes should be computed and stored. For the other boxes the Fast Multipole Method can be used. Now consider one of the boxes at a lower level $\lambda - 1$ denoted by $\mathcal{B}_{n(\lambda-1)}$ contained inside box $\mathcal{B}_{m\lambda}$. That box, which is denoted a son, also has some nearest neighbor boxes, while the other boxes can use the Fast Multipole Method. Since, the boxes at the lower level are smaller some of the boxes that are sons to the nearest neighbor boxes of box $\mathcal{B}_{m\lambda}$ can use the Fast Multipole Method. The idea of the multilevel Fast Multipole Method is to compute the far interactions at the level below the level where they are considered to be near interactions as in Figure 3. As before a tree structure is constructed, but in this case the box size at the lowest level is chosen as small as possible, usually about a quarter of a wavelength for RWG basis functions. At level λ the boxes are numbered $\mathcal{B}_{m\lambda}$ for $1 \leq m \leq M_\lambda$, where M_λ is the number of boxes on level λ. For each box $\mathcal{B}_{m\lambda}$, three lists are constructed. The first list is the nearest neighbor list $\mathcal{I}^{\mathcal{N}}_{m\lambda}$, which is used only at the lowest level after construction of the tree. The second list $\mathcal{I}^{\mathcal{F}}_{m\lambda}$ contains the boxes which can use the Fast Multipole Method on level λ but where the respective fathers are considered to be near neighbors on level $\lambda + 1$. The third list $\mathcal{I}^{\mathcal{S}}_{m\lambda}$ contains the sons of the box or the basis functions on the lowest level.

For each box $\mathcal{B}_{m1}$ on the lowest level 1 the near interactions are computed and stored in $\mathbf{Z}^{near}$, which is defined as before. The far field pattern $\mathbf{F}_l(\hat{\boldsymbol{\kappa}}_{j_1})$ in (11) for each basis function $\mathbf{j}_l$ is also computed for each quadrature point $\hat{\boldsymbol{\kappa}}_{j_1}$, where $1 \leq j_1 \leq K_1$ and K_1 is the number of required integration points on the lowest level. In the same way the receiving pattern $\mathbf{R}_k(\hat{\boldsymbol{\kappa}}_{j_1})$ is computed. The translation operators $\mathcal{T}^L_j(\kappa, \mathbf{X}_{m\lambda} - \mathbf{X}_{n\lambda})$ are computed for all levels. From the far field pattern $\mathbf{F}_l(\hat{\boldsymbol{\kappa}}_{j_1})$ the far field patterns $\tilde{\mathbf{F}}_{m1}(\hat{\boldsymbol{\kappa}}_{j_1})$ of the boxes on the lowest level

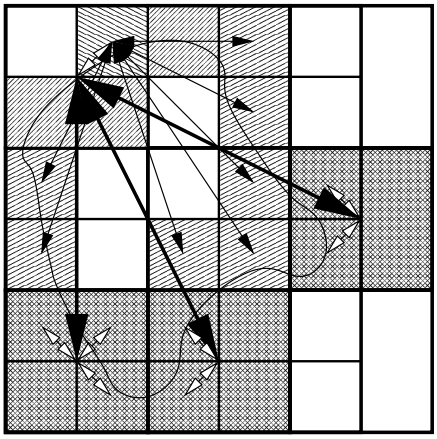

Figure 3: The principle behind the multilevel interactions. Here, the interactions are shown for one box. The nearest neighbors of the box have their interactions computed exactly.

are computed from (11). To compute the far field patterns on the higher levels in an efficient way interpolation and exponential translation from the lower level boxes is used. The key observation is that the contribution from box $\mathcal{B}_{m(\lambda-1)}$ to the father box $\mathcal{B}_{n\lambda}$ is analytically $\tilde{\mathbf{F}}_{n\lambda}(\hat{\boldsymbol{\kappa}}) = e^{\imath\boldsymbol{\kappa}\cdot\left(\mathbf{X}_{n\lambda}-\mathbf{X}_{m(\lambda-1)}\right)}\tilde{\mathbf{F}}_{m(\lambda-1)}(\hat{\boldsymbol{\kappa}})$. Thus, if $\tilde{\mathbf{F}}_{m(\lambda-1)}(\hat{\boldsymbol{\kappa}}_{j_\lambda})$ is known for $1 \leq j_\lambda \leq K_\lambda$ it is easy to compute the far field contribution to the father box, since the computation only involves a multiplication with an exponential, denoted an exponential translation. The operator $\mathcal{P}^{\lambda}_{\lambda-1}(\hat{\boldsymbol{\kappa}}_{j_\lambda})$ is used to denote an interpolation operator that interpolates the discrete far field values on level $\lambda-1$ to the plane wave direction $\hat{\boldsymbol{\kappa}}_{j_\lambda}$ on level λ. Using the interpolation operator one can write $\tilde{\mathbf{F}}_{n\lambda}(\hat{\boldsymbol{\kappa}}_{j_\lambda}) = e^{\imath\boldsymbol{\kappa}_{j_\lambda}\cdot\left(\mathbf{X}_{n\lambda}-\mathbf{X}_{m(\lambda-1)}\right)}\mathcal{P}^{\lambda}_{\lambda-1}(\hat{\boldsymbol{\kappa}}_{j_\lambda})\,\tilde{\mathbf{F}}_{m(\lambda-1)}(\hat{\boldsymbol{\kappa}})$ for $1 \leq j_\lambda \leq K_\lambda$. The far field values of box $\mathcal{B}_{n\lambda}$ is obtained by summing the contributions from its sons.

In the same way the receiving pattern can be interpolated from an equivalent expression. This expression is however not used. Instead the operations are transposed and applied to $\mathbf{G}_{m\lambda}(\hat{\boldsymbol{\kappa}}_{j_\lambda})$ in (12), which leads to a more efficient algorithm. This is sometimes referred to as anterpolation [3]. Equation (13) is computed on the lowest level 1 in a two step recursive way. The first step consists of a sweep from the lowest level and up to the highest level λ_{max}, where all far field patterns on every level are computed from

$$\tilde{\mathbf{F}}_{m\lambda}(\hat{\boldsymbol{\kappa}}_{j_\lambda}) = \begin{cases} \sum_{l\in\mathcal{B}_{m\lambda_{max}}} \mathbf{F}_l\left(\hat{\boldsymbol{\kappa}}_{j_{\lambda_{max}}}\right) I_l & \lambda = 1 \\ \sum_{\mathcal{B}_{n(\lambda-1)}\in\mathcal{I}^{\mathcal{S}}_{m\lambda}} e^{\imath\boldsymbol{\kappa}_{j_\lambda}\cdot\left(\mathbf{X}_{m\lambda}-\mathbf{X}_{n(\lambda-1)}\right)}\mathcal{P}^{\lambda}_{\lambda-1}(\hat{\boldsymbol{\kappa}}_{j_\lambda})\,\tilde{\mathbf{F}}_{n(\lambda-1)}(\hat{\boldsymbol{\kappa}}) & \lambda \neq 1 \end{cases} \tag{14}$$

The second sweep is from the highest level and down, where the translations are

computed and the contributions from the father to the box are added through

$$\mathbf{G}_{m\lambda}\left(\hat{\boldsymbol{\kappa}}_{j_\lambda}\right) = \begin{cases} \sum_{\mathcal{B}_{n\lambda}\in\mathcal{I}^{\mathcal{F}}_{m\lambda}} \mathcal{T}^L_j\left(\kappa, \mathbf{X}_m - \mathbf{X}_n\right)\tilde{\mathbf{F}}_n\left(\hat{\boldsymbol{\kappa}}_{j_\lambda}\right) & \lambda = \lambda_{max} \\ \left(\mathcal{P}^{\lambda+1}_\lambda\left(\hat{\boldsymbol{\kappa}}_{j_\lambda}\right)\right)^T e^{\imath\boldsymbol{\kappa}_{j_\lambda}\cdot\left(\mathbf{X}_{m\lambda}-\mathbf{X}_{n(\lambda+1)}\right)}\mathbf{G}_{n(\lambda+1)}\left(\hat{\boldsymbol{\kappa}}_{j_{(\lambda+1)}}\right) & \\ \qquad + \sum_{\mathcal{B}_{n\lambda}\in\mathcal{I}^{\mathcal{F}}_{m\lambda}} \mathcal{T}^L_j\left(\kappa, \mathbf{X}_m - \mathbf{X}_n\right)\tilde{\mathbf{F}}_n\left(\hat{\boldsymbol{\kappa}}_{j_\lambda}\right) & \lambda \neq \lambda_{max} \end{cases} \tag{15}$$

For the boxes on level 1 the contributions from the far field boxes are calculated from the receiving patterns and $\mathbf{G}_{m1}\left(\hat{\boldsymbol{\kappa}}_{j_1}\right)$ as before in (13). The interpolation can be performed by a sparse matrix. If there is no clustering of the basis functions the number of levels is $\mathcal{O}\left(\log N\right)$. If a sparse interpolation matrix is used the work on each level is proportional to $\mathcal{O}\left(N\right)$ and the total work is $\mathcal{O}\left(N\log N\right)$ [3].

6 Implementation issues

Although the multilevel Fast Multipole Method results in a method that scales as $\mathcal{O}\left(N\log N\right)$, the constant in front of the leading term in the complexity estimate is fairly large. Therefore it is very important to reduce the constant as much as possible. This is especially true for the constant in front of the memory consumption of $\mathcal{O}\left(N\log N\right)$, since it is usually the memory that is the prohibiting factor when solving large problems. The most time consuming parts are the interpolation, the translation and anterpolation parts. Therefore it is crucial to optimize these parts as much as possible. Interpolation and anterpolation are the transpose operators of each other so they benefit from the same optimizations.

6.1 Interpolating the far field pattern

In order to achieve an $\mathcal{O}\left(N\log N\right)$ method the interpolation $\mathcal{P}^\lambda_{\lambda-1}$ between the levels must be fast [3]. Several interpolation schemes exist. Here, the interpolation method in [14] is used. The idea is to interpolate local values from a (θ,ϕ)-grid with N_ϕ nodes in the ϕ-direction and N_θ nodes in the θ-direction to another (θ,ϕ)-grid with $N_{\phi'}$ nodes in the ϕ-direction and $N_{\theta'}$ nodes in the θ-direction. Since the values are located on a sphere the boundary condition for the ϕ-direction is periodic. But since we are only using two components special care must be taken when passing the poles [2]. The simplest approach is to use ordinary Lagrange interpolation, but other interpolation polynomials can be used. To compute $f\left(\theta_{i'},\phi_{k'}\right)$ a number of neighboring points to $(\theta_{i'},\phi_{k'})$ are defined. First the interpolation is carried out in the θ-direction. In each direction, m values with appropriate boundary conditions are selected. Each value

$f(\theta_i, \phi_k)$ is weighted with an interpolation weight ω_i yielding

$$\tilde{f}(\theta_{i'}, \phi_k) = \sum_{i=1}^{m} \omega_i(\theta_{i'}) f(\theta_i, \phi_k) \tag{16}$$

An approximation to $f(\theta_{i'}, \phi_k)$ is then given by $\tilde{f}(\theta_{i'}, \phi_k)$. The coefficients $\omega_i(\theta_{i'})$ can be stored in a sparse matrix $\mathbf{\Omega}_\theta$ of size $N_{\theta'} \times N_\theta$. The interpolation in θ is then defined by the matrix $\mathbf{I}_{N_\phi} \otimes \mathbf{\Omega}_\theta$, where $\mathbf{I}_{N_\phi}$ is the identity matrix of size $N_\phi \times N_\phi$ and $\otimes$ denotes the Kronecker tensor product. In the same way a matrix $\mathbf{\Omega}_\phi \otimes \mathbf{I}_{N_{\theta'}}$ where $\mathbf{\Omega}_\phi$ is a sparse matrix of size $N_{\phi'} \times N_\phi$ is used for interpolation in the ϕ-direction. Using rules for Kronecker tensor products the final interpolation matrix is $\mathbf{W} = \left(\mathbf{\Omega}_\phi \otimes \mathbf{I}_{N_{\theta'}}\right)\left(\mathbf{I}_{N_\phi} \otimes \mathbf{\Omega}_\theta\right) = \mathbf{\Omega}_\phi \otimes \mathbf{\Omega}_\theta$. Note that the final number of entries in the matrix $\mathbf{W}$ is $\mathcal{O}(N_{\phi'} N_{\theta'} m^2)$ while $\mathbf{I}_{N_\phi} \otimes \mathbf{\Omega}_\theta$ has $\mathcal{O}(N_\phi N_{\theta'} m)$ entries and $\mathbf{\Omega}_\phi \otimes \mathbf{I}_{N_{\theta'}}$ has $\mathcal{O}(N_{\phi'} N_{\theta'} m)$ entries. Thus, the work for interpolating with matrix $\mathbf{W}$ is $\mathcal{O}(N_{\phi'} N_{\theta'} m^2)$ while the work when the other matrices are used one at a time is $\mathcal{O}((N_\phi + N_{\phi'}) N_{\theta'} m)$ which is much cheaper for $m \geq 2$, since $N_\phi < N_{\phi'}$. The storage requirement for the second case is only $\mathcal{O}((N_{\phi'} + N_{\theta'}) m)$.

6.2 Reducing the memory for the far field pattern

The far field pattern $\tilde{\mathbf{F}}_{m\lambda}(\hat{\boldsymbol{\kappa}}_{j_\lambda})$ in (14) as well as the translated far field pattern $\mathbf{G}_{m\lambda}(\hat{\boldsymbol{\kappa}}_{j_\lambda})$ in (15) have to be stored. The simplest way is allocate one memory buffer for each of them. A simple method for reducing the memory required was proposed in [5]. It is based on the observation that once the far field pattern on a level is translated to all its far neighbors, it is not needed for any more computations. That memory can be reused by the translated far field patterns on the levels below as in Figure 4. With this approach we need one memory buffer with the size of one of the previous ones plus the size of the memory for the level requiring the most memory, reducing the memory requirement for this part by almost one half.

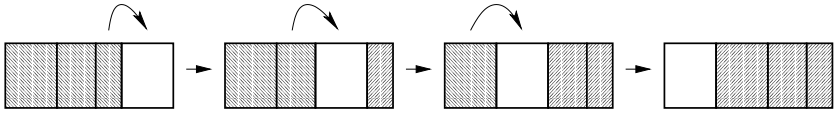

Figure 4: Efficient use of memory for storing the far field patterns.

On the lowest level the far field pattern and receiving pattern for each basis function is needed. A fast implementation of the Fast Multipole method would precompute and store them. One way of reducing the memory requirement is to compute the patterns when they are needed. This gives a memory reduction of $\mathcal{O}(K_1 N)$ but also gives an increase in the computing time of the same order. Thus, it is a trade-off between if it is computational time or memory that is the limiting factor.

6.3 Storing translation operators and far field patterns

A naive way of storing the translation operators $\mathcal{T}_k^L(\kappa, \mathbf{X})$ in (7) is to compute all the translation operators required by a box and store them. This approach misses the crucial point that the translation operators in (7) depend on $\hat{\boldsymbol{\kappa}}$ and $\mathbf{X}$. Since the boxes are inside a grid, the same combination of the values of $\hat{\boldsymbol{\kappa}}$ and $\mathbf{X}$ will appear several times. The best way to store the values is based on the possible values of $\hat{\boldsymbol{\kappa}}$ and $\mathbf{X}$. In a multilevel method there are $(2m+1)^3$ near neighbors of a box. When the boxes are divided in eight smaller boxes on the level below the $(2m+1)^3$ boxes become $(4m+2)^3$ boxes on the lower level. The boxes that are handled using the far field method on the lower level are at most $(4m+2)^3-(2m+1)^3$ boxes. With $m=1$ there are 189 far field boxes. The fact that the boxes on the lower level have a different center compared to their father imply that there are $(4m+3)^3-(2m+1)^3$ possible values on $\mathbf{X}$. For $m=1$ there are 316 possible values. This means that at most $\left((4m+3)^3-(2m+1)^3\right)K_\lambda$ values have to be stored for the translation operator. Using symmetry between different combinations of $\mathbf{X}$ and $\hat{\boldsymbol{\kappa}}$ a significant reduction in the memory storage requirements for the translation operators and also the exponential translation operators in the interpolation is obtained. From symmetry it is actually enough to store $\left((2m+2)^3-(m+1)^3\right)K_\lambda$ translation operator values, i.e. only $56K_\lambda$ values have to be stored for $m=1$.

The directional plane wave vector $\hat{\boldsymbol{\kappa}}$ can be written as $\hat{\boldsymbol{\kappa}} = \sin\theta\cos\phi\hat{\mathbf{x}}_1 + \sin\theta\sin\phi\hat{\mathbf{x}}_2 + \cos\theta\hat{\mathbf{x}}_3$. An ordinary vector $\mathbf{x}$ can be written as $\mathbf{x} = x_1\hat{\mathbf{x}}_1 + x_2\hat{\mathbf{x}}_2 + x_3\hat{\mathbf{x}}_3$. Here, $\mathbf{x}_j$ for $j=1,2,3$ are the three orthogonal unitary vectors in 3-dimensional space. Elementary trigonometry gives the symmetries

$$
\begin{cases}
\hat{\boldsymbol{\kappa}}\cdot(x_1\hat{\mathbf{x}}_1 + x_2\hat{\mathbf{x}}_2 + x_3\hat{\mathbf{x}}_3) = \sin\theta\cos\phi x_1 + \sin\theta\sin\phi x_2 + \cos\theta x_3 \\
\hat{\boldsymbol{\kappa}}\cdot(-x_1\hat{\mathbf{x}}_1 + x_2\hat{\mathbf{x}}_2 + x_3\hat{\mathbf{x}}_3) = \sin\theta\cos(\pi-\phi)x_1 \\
\qquad\qquad + \sin\theta\sin(\pi-\phi)x_2 + \cos\theta x_3 \\
\hat{\boldsymbol{\kappa}}\cdot(x_1\hat{\mathbf{x}}_1 - x_2\hat{\mathbf{x}}_2 + x_3\hat{\mathbf{x}}_3) = \sin\theta\cos(2\pi-\phi)x_1 \\
\qquad\qquad + \sin\theta\sin(2\pi-\phi)x_2 + \cos\theta x_3 \\
\hat{\boldsymbol{\kappa}}\cdot(x_1\hat{\mathbf{x}}_1 + x_2\hat{\mathbf{x}}_2 - x_3\hat{\mathbf{x}}_3) = \sin(\pi-\theta)\cos\phi x_1 \\
\qquad\qquad + \sin(\pi-\theta)\sin\phi x_2 + \cos(\pi-\theta)x_3 \\
\hat{\boldsymbol{\kappa}}\cdot(-x_1\hat{\mathbf{x}}_1 - x_2\hat{\mathbf{x}}_2 + x_3\hat{\mathbf{x}}_3) = \sin\theta\cos(\pi+\phi)x_1 \\
\qquad\qquad + \sin\theta\sin(\pi+\phi)x_2 + \cos\theta x_3 \\
\hat{\boldsymbol{\kappa}}\cdot(-x_1\hat{\mathbf{x}}_1 + x_2\hat{\mathbf{x}}_2 - x_3\hat{\mathbf{x}}_3) = \sin(\pi-\theta)\cos(\pi-\phi)x_1 \\
\qquad\qquad + \sin(\pi-\theta)\sin(\pi-\phi)x_2 + \cos(\pi-\theta)x_3 \\
\hat{\boldsymbol{\kappa}}\cdot(x_1\hat{\mathbf{x}}_1 - x_2\hat{\mathbf{x}}_2 - x_3\hat{\mathbf{x}}_3) = \sin(\pi-\theta)\cos(2\pi-\phi)x_1 \\
\qquad\qquad + \sin(\pi-\theta)\sin(2\pi-\phi)x_2 + \cos(\pi-\theta)x_3 \\
\hat{\boldsymbol{\kappa}}\cdot(-x_1\hat{\mathbf{x}}_1 - x_2\hat{\mathbf{x}}_2 - x_3\hat{\mathbf{x}}_3) = \sin(\pi-\theta)\cos(\pi+\phi)x_1 \\
\qquad\qquad + \sin(\pi-\theta)\sin(\pi+\phi)x_2 + \cos(\pi-\theta)x_3
\end{cases}
$$

(17)

Suppose that a function is of the form $F(\hat{\boldsymbol{\kappa}}, \mathbf{x}) = F(\hat{\boldsymbol{\kappa}} \cdot \mathbf{x})$ and that the function values are known for all possible positive values of x_1, x_2 and x_3 that are needed. If the function values are distributed on a grid of θ and ϕ values for a given $\mathbf{x}$, as in our case, then the symmetries in equation (17) implies that the values in the seven other corners of an octant are known as long as the number of ϕ-values is even. The symmetries (17) are used and implemented in an algorithm in [9]. The algorithm can help compress some of the memory requirements by eight times. Two applications are compression of the translation operator and compression of the exponential translation values. In the case of the exponential translation values the compression is a factor of eight. This is so because the distance from the box to the father is the same for any box on a given level. In the case of the translation operator the factor is lower. If there are m neighboring boxes on a level there are $(2m+2)^3$ possible values for a positive vector $\mathbf{x}$. Since there are $(m+1)^3$ possible values on $\mathbf{x}$ for a near neighbor in the positive octant the number of possible translation operator values is reduced from $\left((4m+3)^3 - (2m+1)^3\right) K_\lambda$ to $\left((2m+2)^3 - (m+1)^3\right) K_\lambda$. For $m = 1$ the compression factor is about 5.6. If the number of ϕ-values is divisible by four then one can use the fact that $\sin(\pi/2 - \alpha) = \cos(\alpha)$ and switch the x_1 and x_2 value to arrive at 8 more symmetries.

For real valued wavenumbers there is another symmetry that can be used to reduce some of the storage requirement by another factor of two [9]. The reduction in this case is due to the relationship $e^{-\imath z} = \overline{e^{\imath z}}$ when z is real. For the exponential translation values in (14) this can be used to reduce the storage requirement by a factor of two. It can also be used to reduce the storage requirement for the far field and receiving patterns. For the far field pattern we have $\mathbf{F}_l(-\hat{\boldsymbol{\kappa}}) = \overline{\mathbf{F}_l(\hat{\boldsymbol{\kappa}})}$. Note that if $\hat{\boldsymbol{\kappa}}$ is a quadrature point then so is $-\hat{\boldsymbol{\kappa}}$. Thus, only one of the far field values have to be stored.

The symmetry for the receiving pattern in (9) is not as obvious. However, if the symmetry relation $\mathbf{R}_k(-\hat{\boldsymbol{\kappa}}) - \alpha\overline{\mathbf{F}_k(-\hat{\boldsymbol{\kappa}})} = -\left(\overline{\mathbf{R}_k(\hat{\boldsymbol{\kappa}}) - \alpha\overline{\mathbf{F}_k(\hat{\boldsymbol{\kappa}})}}\right)$ is used and $\mathbf{K}_k(\hat{\boldsymbol{\kappa}}) = \mathbf{R}_k(\hat{\boldsymbol{\kappa}}) - \alpha\overline{\mathbf{F}_k(\hat{\boldsymbol{\kappa}})}$ is stored, one can compute $\mathbf{R}_k(\hat{\boldsymbol{\kappa}})$ from $\mathbf{K}_k(\hat{\boldsymbol{\kappa}})$ and $\mathbf{F}_k(\hat{\boldsymbol{\kappa}})$. Since the far field pattern is already stored the storage requirements are reduced by a factor of two. If the far field and receiving patterns are computed when they are needed to reduce the memory requirements a speedup factor of almost two is obtained when this symmetry is used.

6.4 Memory reduction from box manipulations

From experience we have learned that most of the memory consumption is due to the storage of the near field matrix $\mathbf{Z}^{near}$ and the storage of the far field and receiving patterns on the lowest level. This is also the conclusion in [3].

It is therefore critical that this storage is kept as low as possible, especially for large objects where memory becomes the limiting factor. On the lowest level the boxes usually have a prescribed smallest diameter. This diameter is not always attained since the object is enclosed in a bounding box and subdivided until one more subdivision results in a box that is too small. The problem is that a larger box diameter leads to more entries in the near field matrix and more quadrature points for the far field and receiving patterns. In [6] a technique that is based on rotating an object in order to reduce the bounding box is introduced. Object rotation allows a reduction of the bounding box if at least one of the length scales is much smaller than the other two. Since the bounding box is smaller, the diameter of the box on the lowest level is also reduced assuming that the diameter of the bounding box has not shrunk too much. For uniformly distributed objects this method does not give any memory gain since the bounding box is not affected by rotations.

We propose another method that can be used to reduce the box diameter. In this method the diameter of the boxes on the lowest level is forced to be the prescribed smallest diameter. The box diameter is then doubled until the object is enclosed in one bounding box. This bounding box will usually be larger than the original bounding box. The object is then placed in a corner of the bounding box. This will ensure that the minimum number of boxes on each level is used. If the object is centered in the bounding box there is a risk that the boxes on the lowest level force the use of more boxes on the higher levels. This would be the case if at a given level the used boxes are surrounded by a layer of unused boxes in all directions. An example is given in Figure 5. If the bounding box is larger and the diameter of the boxes on the lowest level is smaller there will usually be an extra level. A problem is to choose which level that should be the highest level λ_{max}. Choosing the highest possible level might not be optimal in terms of arithmetic operations. Since the object is placed in a corner the boxes on the highest level might have few far interactions. Also, because the boxes on the highest level are larger than the boxes in the original case, more quadrature points are needed. Therefore, it might require more work to interpolate and translate and anterpolate on the highest level than to carry out translations between the sons on one level below directly.

Let us now quantify the reductions. Assume that an object has no clustering points. Let the side length of the original box be d_1 and the new box be d_2, then $d_2 \leq d_1 < 2d_2$. Let m be the number of near neighbor boxes in any direction. Since the object is a surface, each basis function interacts with $C\left(2m+1\right)^2 d_1^2$ other basis functions in the original case and $C\left(2m+1\right)^2 d_2^2$ in the second case, where C is a constant depending on the discretization density. Thus, the near field is reduced to $\left(d_2/d_1\right)^2$ of the original size. In the extreme case this is almost as low as 25 % of the original size. The reduction in the far field can be of the same order, but is usually not. The number of quadrature points for

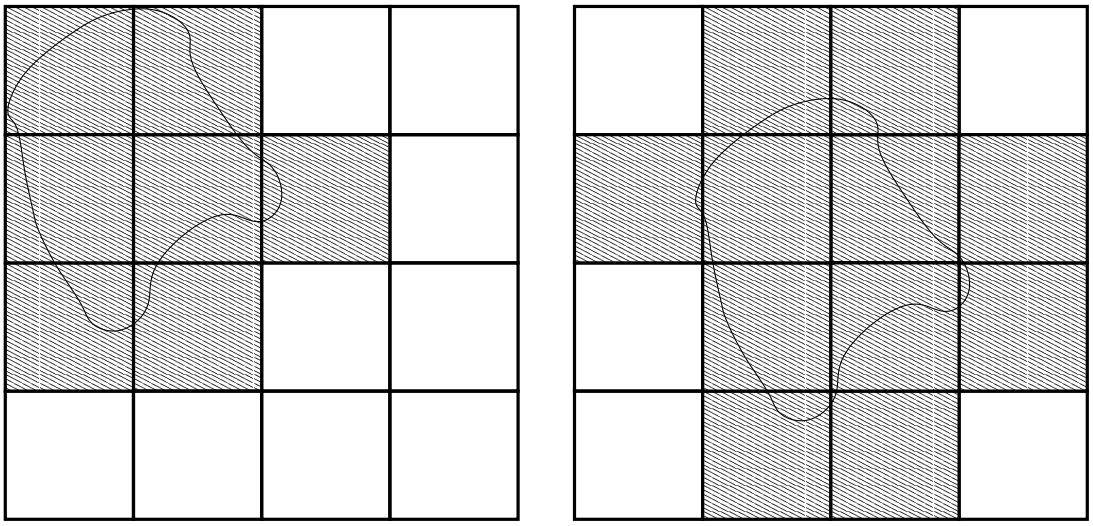

Figure 5: The difference between putting the object in the corner of the bounding box (left) and centering the object in the bounding box (right) when the box size is forced. In this case the number of boxes that encloses the object on the given level is 4×4. When the object is put in the corner the number of boxes stored are 7 and the number of far away interactions are at most 3. In the centered case there are 11 boxes and at most 7 far away interactions.

each basis function in the original case is $2\left(\sqrt{3}\kappa d_1 + \beta\left(\sqrt{3}\kappa d_1\right)^{\frac{1}{3}} + 1\right)^2$ and is $2\left(\sqrt{3}\kappa d_2 + \beta\left(\sqrt{3}\kappa d_2\right)^{\frac{1}{3}} + 1\right)^2$ in the second, cf. (4). When $\beta \ll (\kappa d_2)^{-\frac{2}{3}}$ the reduction is $(d_2/d_1)^2$ of the original size and when $(\kappa d_1)^{-\frac{2}{3}} \ll \beta$ the reduction is $(d_2/d_1)^{\frac{2}{3}}$. In the first case the memory can be reduced to as low as 25 % of the original size. In the second case the memory can be reduced to as low as 63 % of the original size. Since d_1 and d_2 are chosen as small as possible we are often closer to the second case.

7 Shared memory parallelization

Here, we develop a parallelization strategy for shared memory computers running OpenMP [10]. A shared memory computer is a multiple processor machine with one memory that can be accessed from all processors. The processors are often arranged so that each processor has a local memory and a fast interconnect to the local memory of the other processors. Thus, there is a slight delay if an access is attempted at a memory segment that is not in cache or the local memory.

In the Fast Multipole Method for CFIE the work on each level is approximately constant [3], but the work on each level is distributed differently. On the

lower levels there are many boxes, but because of the small box size the number of quadrature points are relatively few. On the higher levels it is the opposite way, the number of boxes are relatively few but the box size is large so the number of quadrature points is large. The dominant cost comes from interpolation and anterpolation and translation. It is therefore important that these parts are well parallelized. A simple minded strategy for shared memory parallelization of a Fast Multipole code is to parallelize the loops over boxes, as in Figure 6, since it is the outermost loop. This is easy to implement with OpenMP, because it only requires a single parallelization directive. For the near field and far field pattern and radiation pattern on level 1 there are many boxes which means that the simple strategy to parallelize over the boxes is sufficient for this part. If the boxes have an approximately equal number of basis functions each processor will have an approximately equal workload of

$$T_1(p) = t_1 \left\lceil \frac{M_1}{p} \right\rceil \tag{18}$$

Here, $\lceil x \rceil$ denotes the smallest integer greater than or equal to x, t_1 is the average serial processor time for each box, M_1 is the number of boxes on level 1 and p is the number of processors.

For the interpolation and anterpolation and translation this strategy is efficient only for the lower level boxes, since there are many boxes while there is only a small number of quadrature points. Because of the Morton ordering, boxes that are physically close are also close in the box list. If the box list is distributed evenly between the processors, boxes that are physically close are likely to belong to the same processor. Thus, most memory accesses are local in this case. The memory accesses that are non-local in this case are mainly due to the translation part. This leads to an execution time

$$T_\lambda(p) = t_\lambda \left\lceil \frac{M_\lambda}{p} \right\rceil K_\lambda + c_\lambda N_\lambda(p) K_\lambda + u_\lambda N_\lambda(p) \tag{19}$$

for each level. Here, t_λ is the average serial processor time for each box on level λ, c_λ is the communication time for one quadrature point value and u_λ is the communication upstart time for each box, M_λ is the number of boxes on level λ and K_λ is the number of quadrature points, $N_\lambda(p)$ is the average number of non-local boxes for each box, which is a number that depends on the number of processors. For higher level boxes there are two problems with this method. The boxes are few while the number of quadrature points is large. Therefore most memory accesses are non-local. For geometric reasons the number of quadrature points on the highest level is $\mathcal{O}(N)$. Thus, the accesses to non-local data takes considerable time. When $p > M_\lambda$ some processors are idle. Thus, the scaling $T_\lambda(1)/T_\lambda(p)$ cannot be better than M_λ.

Because of the above mentioned problems a different strategy is needed. First note that the total work on any level is $t_\lambda M_\lambda K_\lambda$. The previous strategy was to

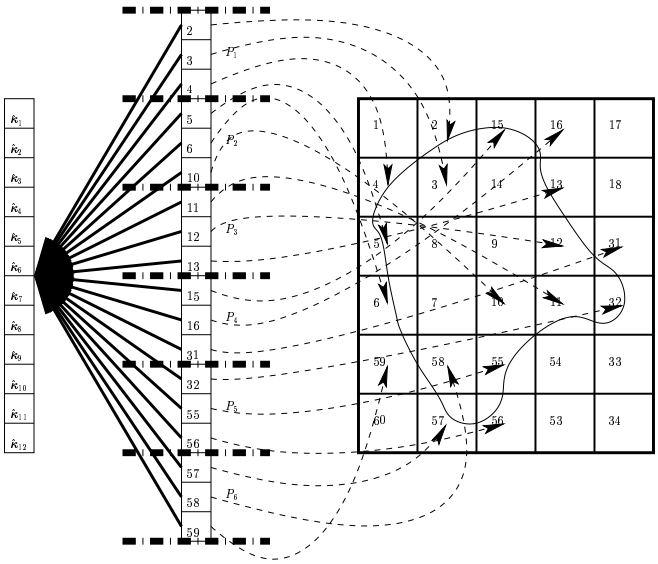

Figure 6: Each processor is assigned a number of boxes and the operations are carried out over all quadrature points.

divide the boxes into p equal pieces. For large values of K_λ and small values of M_λ it seems better to divide the quadrature points instead, as in Figure 7. This method was proposed in [15]. Since the translation operator is diagonal there is no communication cost for it. Instead there is an added cost for the interpolation and anterpolation. The added cost is mainly due to non-local memory accesses. When the level below is parallelized in the same way, non-local memory accesses occur for the quadrature points in the θ-direction that need quadrature points placed on another processor to complete its interpolation. This occurs for the quadrature points on the (θ, ϕ)-grid on the border between two processors. This results in that the the non-local accesses are in the θ-direction. The number of non-local memory accesses is approximately $M_\lambda m\sqrt{2K_{(\lambda-1)}}$, where m is the number of interpolation points. The reason is that the number of non-local memory accesses depends on the number of points in the interpolation and the number of quadrature points in the ϕ-direction on the level below. The total cost is then

$$T_\lambda(p) = t_\lambda M_\lambda \left\lceil \frac{K_\lambda}{p} \right\rceil + c_\lambda M_\lambda m\sqrt{2K_{(\lambda-1)}} + u_\lambda M_\lambda \tag{20}$$

The previous analysis was based on the assumption that the same method was used on the level below. When the transition is made from one of the methods to the other it does not hold. In this case the communication cost is higher. In order to make a smooth transition from one of the methods to the other we propose a hybridization between the two techniques. In the hybrid method, the

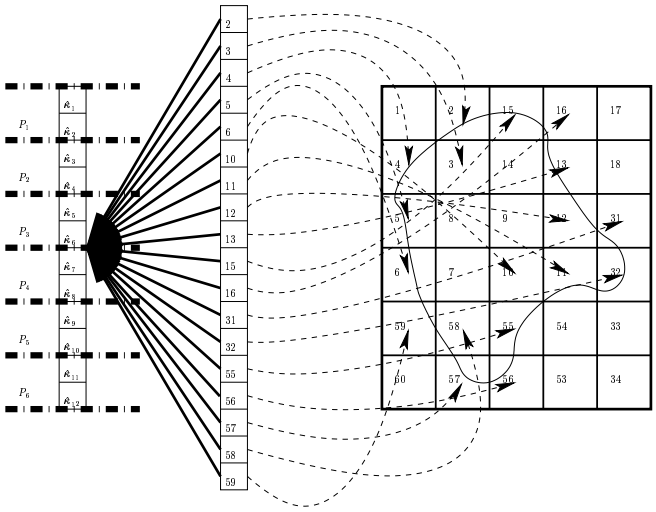

Figure 7: Each processor is assigned a number of quadrature points and carry out the operations on all boxes.

processors are blocked together in $\lceil p/b_\lambda \rceil$ blocks of size b_λ. Then the boxes on level λ are blocked according to the block size b_λ. For each block of boxes the processor block is distributed with the second method, as in Figure 8. The final result is a method with the total cost

$$\begin{aligned} T_\lambda(p) = t_\lambda \left\lceil \frac{M_\lambda}{b_\lambda} \right\rceil \left\lceil \frac{K_\lambda}{\left\lceil \frac{p}{b_\lambda} \right\rceil} \right\rceil + c_\lambda N_\lambda \left(\left\lceil \frac{p}{b_\lambda} \right\rceil \right) \left\lceil \frac{K_\lambda}{\left\lceil \frac{p}{b_\lambda} \right\rceil} \right\rceil \\ + c_\lambda \left\lceil \frac{M_\lambda}{b_\lambda} \right\rceil m\sqrt{2K_{(\lambda-1)}} + u_\lambda \left(N_\lambda \left(\left\lceil \frac{p}{b_\lambda} \right\rceil \right) + \left\lceil \frac{M_\lambda}{b_\lambda} \right\rceil \right) \end{aligned} \tag{21}$$

We note that there will usually be an optimal choice of b_λ, which is computer and implementation dependent. The choice can also depend on the geometry used.

In the experiments we will use different parallelization strategies on different levels in order to get an indication of which of (19) and (20) and (21) that is fastest. As we expect, it turns out that for the lower levels the box strategy is the best. For the highest level the parallelization over quadrature points is the most efficient. The hybrid method is most efficient to use in between.

8 Experiments

The code implementation used for the experiments is a FORTRAN 90 code parallelized with OpenMP. It uses double precision arithmetic. The parallelization can be performed in different ways as in Section 7. The code runs on several platforms and has been validated several times, see for instance [8, 9] and references

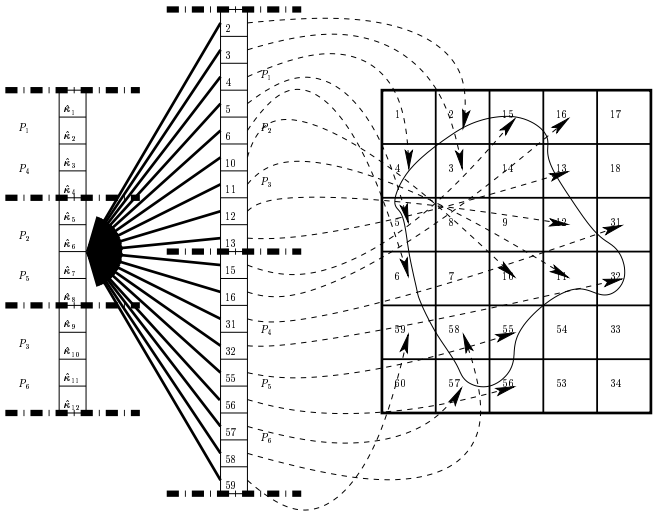

Figure 8: Hybrid parallelization over boxes and quadrature points.

therein. In these experiments the focus is on parallel and serial performance. The computer used is a SUN Fire 15k with 900 MHz processors with a peak performance of 1800 Mflops and 1 Gb of shared memory on each processor.

To test the serial performance and scaling with respect to the number of unknowns we choose the problem of scattering from spheres with radius 1 m and different discretization densities as in [3, 7]. As in [7] the number of multipoles is chosen as $L = \kappa d + (\kappa d)^{\frac{1}{3}}$ and as in [3] four interpolation points are used in the Lagrange interpolation. It should be noted that this gives low accuracy in the Fast Multipole method. Data on the spheres are given in Table 1. As a rule of thumb the frequencies have been chosen to approximately keep the edge length at 10 points per wavelength.

Table 1: Definition of test cases.

Test	Nr. unkn.	Max. edge (m)	Min. edge (m)	Freq. (MHz)
1	3000	0.1319	0.09435	240
2	20280	0.05087	0.03512	660
3	60750	0.02940	0.02012	1100
4	151230	0.01864	0.01269	1800
5	1004670	0.007230	0.004902	4600

In Figure 9 the execution time for one matrix vector product as a function of problem size is plotted. For comparison we also plot the lines $10^{-5} N \log_{10} N$ and $3 \cdot 10^{-5} N \log_{10} N$. Several different configurations are tested. We compare the efficiency of computing far field patterns when they are needed as in Section 6.2, denoted computed, with the standard method when they are stored in memory, denoted stored. For each method the result of the box manipulations

in Section 6.4, denoted forced, is compared to the standard case. We also check if there is any added benefit in reducing the highest level, denoted -1 level, as was suggested in Section 6.4. When the problem is small there is a slight benefit in using the standard box diameter but this advantage diminishes for larger problems. For larger problems the reduced memory accesses compensates for the slightly larger cost for matrix vector multiplication. On the other hand if the far field is computed when it is needed the box manipulation pays off since a large part of the computation is spent in this part, as was noted in Section 6.2. So, in this case forcing the box diameter to be the smallest possible is mandatory. Reducing the level by one gives comparable results with the forced case.

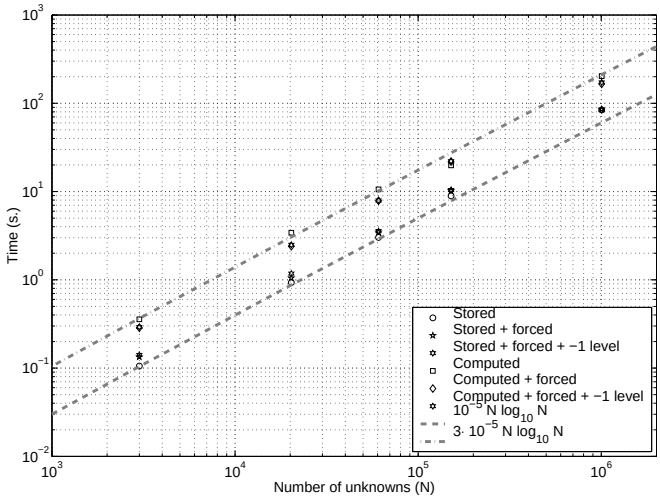

Figure 9: The time for one matrix vector multiplication by the Fast Multipole method as a function of number of unknowns.

In Figure 10 we compare the memory consumption for the different cases. The lines $4 \cdot 10^{-3}N$ and $7 \cdot 10^{-3}N$ are plotted for comparison. The memory usage is the one reported by the system, which includes memory used by the program, the iterative solver and the geometry in addition to the Fast Multipole method. Compared to the standard method there is a reduction when the far field is computed when needed, but it is actually less than when the box diameter is forced to the smallest diameter instead. The combination of the two methods reduces the memory consumption to 57 % in the largest case. Thus, the first action when the problem is too large to fit into memory is to force the diameter of the smallest box. The second action is to compute the far field patterns when they are needed.

The theoretical estimates for the memory reduction in Section 6.4 are investigated in Figure 10. For test case 4 in Table 1 there is no reduction in the far field memory despite the theoretical prediction. The reason is that the theory is based on the assumption that the number of multipoles is a continuous function of box diameter, which is not the case. The reduction of the box diameter is not

sufficient to reduce the number of multipoles on the lowest level. In this case, the box diameter is close to optimal without any reduction. The figure indicates that the theoretical assumptions are reasonable.

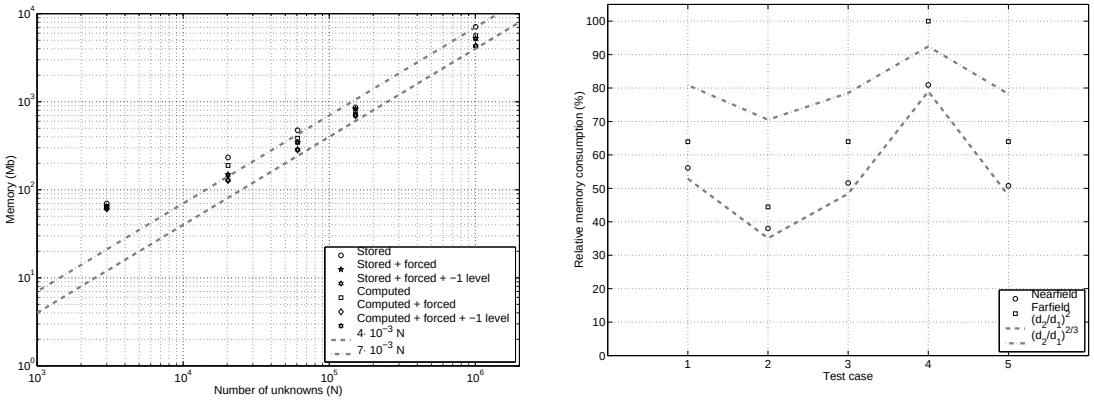

Figure 10: The memory usage as a function of number of unknowns (left). The memory consumption relative to the standard box size when the smallest box diameter is forced (right).

The parallel performance on the sphere is investigated in Figure 11 and Figure 12. Here, $\beta = 1.0$ is used to compute the number of multipoles. In Figure 11 test case 4 is used and in Figure 12 test case 5 in Table 1 is used. The speedup $T(1)/T(p)$ and parallel efficiency $T(1)/(pT(p))$ for the matrix vector multiplication is shown. The different methods in Section 7 are tested. The simple method with parallelization over boxes is compared to three cases. The first one is when the highest level is parallelized over quadrature points. The second one is when the two highest levels are parallelized over quadrature points. The third case is parallelized over quadrature points on the highest level and use the hybrid method with $b_\lambda = 2$ in (21) on the second highest level.

For test case 4 the maximum speedup is 14.7 obtained with 26 processors and the method that use the hybrid method on the second highest level. For this case the parallelization over boxes is only slightly slower than the other three methods. The methods parallelized over quadrature points performs approximately the same, indicating that there is a number of levels that gives approximately the same performance. Choosing the block size $b_\lambda = 2$ in (21) on the second highest level gives the best performance for the methods tested.

For test case 5 the maximum speedup is 21.5 on 26 processors again obtained with the method that use the hybrid method on the second highest level. Here, the speedup for the parallelization over boxes is approximately the same as in the previous case. Increasing the problem size does not seem to improve the parallel performance for this case. On the other hand all the other cases show improved performance compared to the smaller case. Also, the hybrid method is clearly the best choice in this case managing to give a parallel efficiency of over 80 %

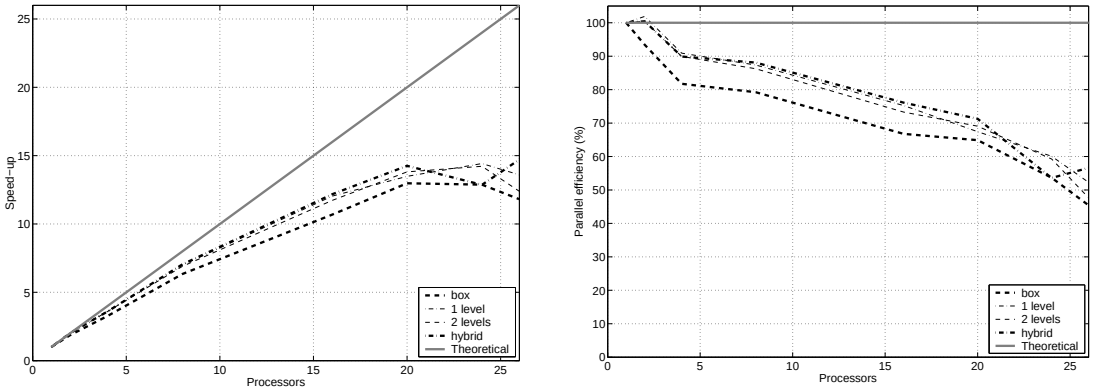

Figure 11: The speedup (left) and the efficiency (right) for the problem with 151230 unknowns.

with 26 processors. As in the previous case the two methods with parallelization over quadrature points are placed between the other two. For larger problems

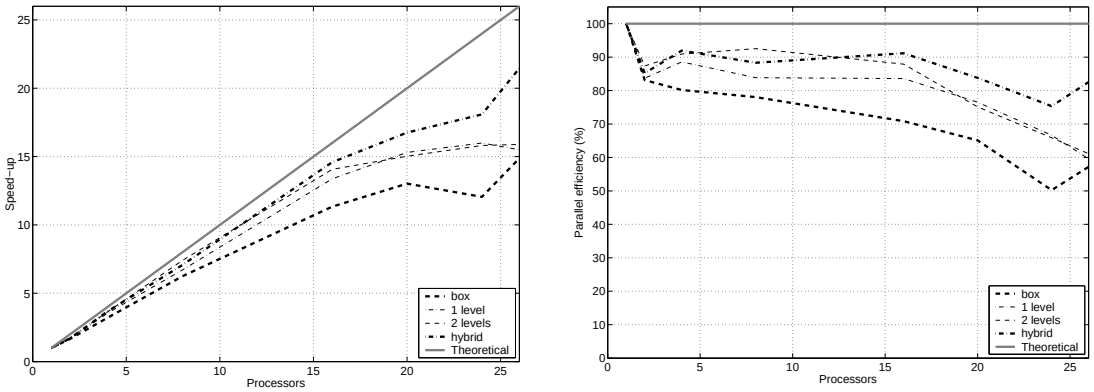

Figure 12: The speedup (left) and the efficiency (right) for the problem with 1 million unknowns.

the hybrid method shows greater potential than the other methods. For both cases the parallelization over boxes is insufficient. The three other methods perform very well up to 16 processors giving more than 73 % of the single processor performance in the worst case.

Finally, we also compare the parallelization over boxes with the previous hybrid method for a model of the full scale jet fighter SAAB Trainer in Figure 13. The fighter is 11 m. long and is discretized with 191712 unknowns at 1 GHz.

In Figure 14 the speedup and parallel efficiency are shown. Also in this case the hybrid method performs better than the parallelization over boxes. The maximum speedup is 15.2 in this case. The results are similar to the ones for test case 4, as expected since the problem sizes are approximately the same.

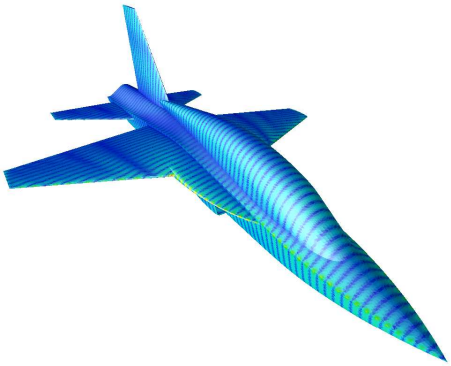

Figure 13: The induced surface currents at 1 GHz on the jet fighter SAAB Trainer.

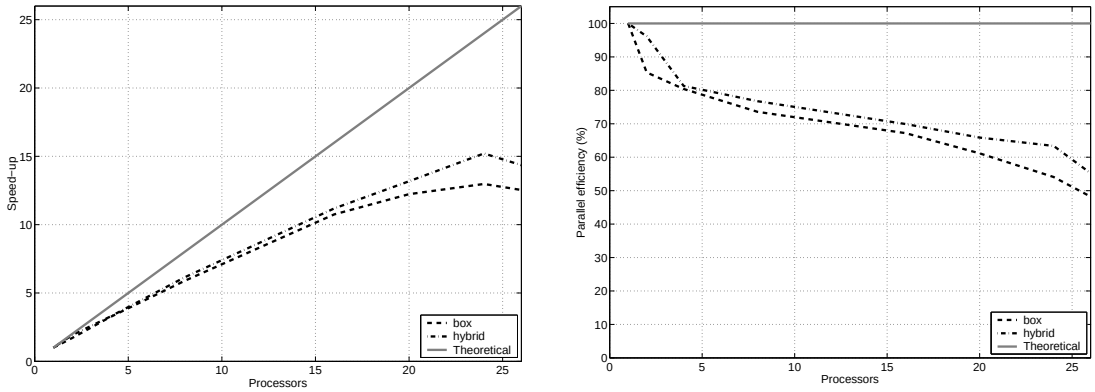

Figure 14: The speedup (left) and the efficiency (right) for the SAAB Trainer.

9 Conclusions

In this paper, an implementation of the Fast Multipole method for electromagnetics was presented. Several issues related to memory consumption and efficiency were dealt with. A new method consisting of a hybrid between previous methods was proposed in order to parallelize efficiently on shared memory multiprocessor machines. The new method scales better than the previous ones.

Acknowledgment

The author would like to thank Prof. Per Lötstedt at Uppsala University for his helpful comments and Jonas Hamberg at SAAB Aerospace for providing the geometry SAAB Trainer.

References

[1] George B. Arfken and Hans J. Weber. *Mathematical Methods for Physicists*. Academic Press, Inc., 4th edition, 1995.

[2] Ovidio M. Bucci, Claudio Gennarelli, and Catello Savarese. Optimal interpolation of radiated fields over a sphere. *IEEE Transactions on Antennas and Propagation*, 39(11):1633–1643, November 1991.

[3] Weng Cho Chew, Jian-Ming Jin, Eric Michielssen, and Jiming Song. *Fast and Efficient Algorithms in Computational Electromagnetics*. Artech House, Inc., Norwood, 2001.

[4] Ronald Coifman, Vladimir Rokhlin, and Stephen Wandzura. The fast multipole method for the wave equation: A pedestrian prescription. *IEEE Transactions on Antennas and Propagation*, 35(3):7–12, June 1993.

[5] Eric Darve. The fast multipole method: Numerical implementation. *Journal of Computational Physics*, 160(1):195–240, 2000.

[6] Michael Larkin Hastriter and Weng Cho Chew. Memory reduction in mlfma through target rotation. In *Proceedings of Antennas and Propagation Society International Symposium*, volume 2, pages 306–309, Columbus, Ohio, USA, June 2003.

[7] Bin Hu, Weng Cho Chew, and Sanjay Velamparambil. Fast inhomogeneous plane wave algorithm for the analysis of electromagnetic scattering. *Radio Science*, 36(6):1327–1340, November/December 2001.

[8] Per Lötstedt and Martin Nilsson. A minimum residual interpolation method for linear equations with multiple right hand sides. Technical Report 2002-041, Department of Information Technology, Scientific Computing, Uppsala University, Dec. 2002. Submitted to SIAM J. Sci. Comp. Available at: http://www.it.uu.se/research/reports/2002-041/.

[9] Martin Nilsson. *Iterative solution of Maxwell's equations in frequency domain*. Licentiate thesis No. 2002-004, Department of Information Technology, Uppsala University, May 2002. Available at: http://www.it.uu.se/research/reports/lic/2002-004/.

[10] OpenMP Architecture Review Board. *OpenMP Fortran Application Program Interface, Version 2.0*, November 2000. Available at: www.openmp.org (2003-09-08).

[11] Andrew F. Peterson, Scott L. Ray, and Raj Mittra. *Computational Methods for Electromagnetics*. IEEE Press and Oxford University Press, New York, Oxford, Tokyo, Melbourne, 1998.

[12] Sadasiva. M. Rao, Donald. R. Wilton, and Allen. W. Glisson. Electromagnetic scattering by surfaces of arbitrary shape. *IEEE Transactions on Antennas and Propagation*, 30(3):409–418, May 1982.

[13] Vladimir Rokhlin. Diagonal forms of translation operators for the Helmholtz equation in three dimensions. *Applied and Computational Harmonic Analysis*, 1(1):82–93, 1993.

[14] Jiming Song and Weng Cho Chew. Multilevel fast multipole algorithm for solving combined field integral equation of electromagnetic scattering. *Microwave and Optical Technology Letters*, 10(1):14–19, September 1995.

[15] Sanjay Velamparambil, Weng Cho Chew, and Michael Larkin Hastriter. Scalable electromagnetic scattering computations. In *Proceedings of Antennas and Propagation Society International Symposium*, volume 3, pages 176–179, San Antonio, Texas, USA, June 2002.

[16] Michael S. Warren and John K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87(1–2):266–290, May 1995.