Comparison of the efficiency of translation operators used in the fast multipole method for the 3D Laplace equation

Nail A. Gumerov and Ramani Duraiswami

April 12, 2005

Abstract

We examine the practical implementation of a fast multipole method algorithm for the rapid summation of Laplace multipoles. Several translation operators with different asymptotic computational and memory complexities have been proposed for this problem. These algorithms include: Method 0 – the originally proposed matrix based translations due to Greengard and Rokhlin (1987), Method 1 – the rotation, axial translation and rotation algorithm due to White and Martin Head-Gordon (1993), and Method 3 – the plane-wave version of the multipole-to-local translation operator due to Greengard and Rokhlin (1997). We compare the algorithms on data sets of varying size and with varying imposed accuracy requirements. While from the literature it would have been expected that method 2 would always be the method of choice, at least as far as computational speed is concerned, we find that this is not always the case. We find that as far as speed is concerned the choice between methods 1 and 2 depends on problem size and error requirements. Method 2 is the algorithm of choice for large problems where high accuracy is required, though the advantage is not clear cut, especially if memory requirements are an issue. If memory is an issue, Method 1 is the method of choice for most problems. A new analysis of the computational complexities of the algorithms is provided, which explains the observed results. We provide guidelines for choosing parameters for FMM algorithms.

1 Introduction

Many problems in electrostatics, fluid mechanics, molecular dynamics, stellar dynamics, etc., can be reduced to evaluation of the field due to a large number of monopole or multipole sources located at locations $\mathbf{x}_\alpha$, $\alpha = 1 \ldots, N$.. For such problems, usually, the most expensive computational part is related to performing a matrix-vector multiplication or summation of the type

$$v(\mathbf{y}_j) = \sum_{i=1}^{N} u_i \Phi(\mathbf{y}_j - \mathbf{x}_i), \quad j = 1, ..., M \tag{1}$$

where $\Phi(\mathbf{y} - \mathbf{x})$ is some function (e.g., the Green's function or multipole solution for the Laplace equation) centered at $\mathbf{x}$, which must be evaluated at locations $\mathbf{y}_j$. Here u_i are some coefficients (e.g. the intensities of the monopole or multipole sources). Straightforward computation of these sums, which also can be considered to be the multiplication of a $M \times N$ matrix with elements $\Phi_{ji} = \Phi(\mathbf{y}_j - \mathbf{x}_i)$ by a N vector with components u_i to obtain a M vector with components $v_j = v(\mathbf{y}_j)$, obviously requires $O(MN)$ operations. The point sets $\mathbf{y}_j$ and $\mathbf{x}_i$ in these problems may be different, or the same. If the points $\mathbf{y}_j$ and $\mathbf{x}_i$ coincide, the evaluation of Φ must be appropriately regularized (e.g., in a boundary element application, quadrature over the element will regularize the function). In the sequel we assume that this issue, if it arises, is dealt with.

The Fast Multipole Method, introduced by Greengard and Rokhlin [Greengard87], seeks to speed up the matrix-vector product by performing it approximately, but with a guaranteed specified accuracy. The main idea in the FMM is to split the sum into a near and a far field

$$v\left(\mathbf{y}_j\right) = \sum_{\mathbf{x}_i \in R_{near}} u_i \Phi\left(\mathbf{y}_j - \mathbf{x}_i\right) + \sum_{\mathbf{x}_i \in R_{far}} u_i \Phi\left(\mathbf{y}_j - \mathbf{x}_i\right), \tag{2}$$

and build factored approximate representations of the functions Φ in the far-field

$$\Phi\left(\mathbf{y}_j - \mathbf{x}_i\right) = \sum_{l=0}^{P-1} S_l\left(\mathbf{y}_j - \mathbf{x}_*\right) R_l\left(\mathbf{x}_i - \mathbf{x}_*\right) + O\left(\epsilon\right), \tag{3}$$

which usually come from analytical series representations, and are truncated at some number of coefficients P, which is a function of the desired precision $\epsilon\left(=\epsilon\left(P\right)\right)$. These factorizations are usually in terms of a singular set of multipole solutions S_l, and a regular set of solutions R_l. These factored parts allow one to separate the computations involving the points sets $\{\mathbf{x}_i\}$ and $\{\mathbf{y}_j\}$, and consolidate operations for many points.

$$\sum_{\mathbf{x}_i \in R_{far}} u_i \Phi\left(\mathbf{y}_j - \mathbf{x}_i\right) = \sum_{\mathbf{x}_i \in R_{far}} u_i \sum_{l=0}^{P-1} S_l\left(\mathbf{y}_j - \mathbf{x}_*\right) R_l\left(\mathbf{x}_i - \mathbf{x}_*\right) \tag{4}$$

$$= \sum_{l=0}^{P-1} S_l\left(\mathbf{y}_j - \mathbf{x}_*\right) \sum_{\mathbf{x}_i \in R_{far}} u_i R_l\left(\mathbf{x}_i - \mathbf{x}_*\right) = \sum_{l=0}^{P-1} C_l S_l\left(\mathbf{y}_j - \mathbf{x}_*\right) \tag{5}$$

The coefficients C_l for all $\mathbf{x}_i$ are built and then used in the evaluation at all $\mathbf{y}_j$. This reduces the cost of this part of the summation to linear order, and reduces the memory requirements to this order as well.

No global factorization (that is quickly convergent) is available, and the split between the near and far-fields must be managed on a part by part basis. This is done by using appropriate data structures, and a variety of representations are used. Due to the dependence of the error on the number of points, the cost of building data-structures that allow this factor, effectively the algorithm achieves a complexity of $O(M \log M + N \log N)$,which can be substantially faster than the direct product for large M and N.

Thus, the FMM is an approximate method, and a feature of its analysis and implementation is the evaluation of the errors. One should not be misled into thinking that the FMM is not accurate. If desired, even machine precision can be achieved, making the solution indistinguishable from an "exact" algorithm. Since its introduction in the late 1980s, the fast multipole method (FMM) has been hailed as one of the top ten algorithms of the 20th century. FMM inspired algorithms have appeared for the solution of various problems of both matrices associated with the Laplace potential, and with those of other equations (the biharmonic, Helmholtz, Maxwell) and in unrelated areas (for general radial basis functions).

When one wishes to implement an FMM algorithm from the material presented in the literature, one is faced with several choices, and the situations where each of these choices should be the correct one is not clear. The purpose of this paper is to describe these choices for the potentials of the Laplace equation, and then present the results of the tests on algorithms implemented with these choices, in terms of their accuracy, their speed of computation, the memory required, and their ease of implementation. Analysis that helps explain the results is also provided.

1.1 Scope of this paper

The issues we consider in this paper are

Translation operators: In the FMM we build various representations of the functions (these will be described below). To convert from one representation in a particular coordinate frame, to the same, or different, representation centered in another coordinate frame, one must "translate" the representation. A typical representation involves p^2 coefficients. In the original FMM [Greengard87] the translation operators were $p^2 \times p^2$ matrices, that acted on the coefficients of one representation to provide the coefficients in the other, at a cost of $O(p^4)$ operations per translation. White and Head-Gordon [White96] provided translation operators that used symmetries in the representing functions along certain the axial coordinate directions, to achieve the translation via a sequence of rotation, translation and rotation operations, which require $O(p^3)$ operations per translation. Greengard and Rokhlin [Greengard97] presented a method that reduced the cost of the most costly of the translation operations to $O\left(p^2\right)$,though the process of converting the functions to this representation remained at $O(p^3)$, and the other translations follow the $O(p^3)$ or $O(p^4)$ translation method. This method also requires more memory resources than for the other translation operators.

Desired accuracy: Often the FMM is used in a certain context, where there are certain errors inherent in the remaining numerical procedures (e.g., in representation of surfaces or boundaries, in the values of experimentally measured data, etc.) and the FMM should be accurate enough that it does not worsen the error in the computation, but should not unnecessarily expend resources in seeking to achieve accuracy that is more than necessary. Corresponding to each desired accuracy, the translation methods will have different values of p. A systematic comparison of the values of p (and as a consequence the desired error ϵ) at which each of these translation methods is most effective, has to our knowledge not been presented (the original papers just compare the results with the direct matrix-vector product). This paper attempts to fill this lacuna in the literature.

Memory resources: One of the FMM's big promises arises due to its requirement of order of magnitude smaller memory. This should allow simulations to be effectively run on the desktop computers of scientists. As such in many applications the memory requirements may be the determining factor in the choice of an algorithm. We accordingly compare the use of the memory in FMM algorithms with varying problem size, error requirements, and memory budgets. An interesting aspect of our experience was that for many problems, larger memory leads to larger time on the commodity architectures we considered.

1.2 Other fast algorithms not considered

One can find several other $O(N)$ or $O(N \log N)$ methods for solution of problem (1), that are not considered here mainly because they either do not appear efficient enough, or cannot achieve the necessary accuracy. We briefly discuss these methods, and why we do not consider them.

The first of these are "tree-code" methods due to Barnes and Hut [Barnes86]. This algorithm was introduced a year or so earlier than the FMM and currently is used in astrophysics [Dehnen02] and other areas e.g., vortex dynamics, [Lindsay01]. In its current version, this method shows high performance for relatively low accuracy simulations, and in fact may be the most efficient method for some problems. The speed of the method relative to the FMM reduces as higher accuracy is required. This is related to the need to use smaller "opening angles" - the parameter, which controls separation of the group of points. We did some preliminary tests, which show that for low accuracy the Barnes and Hut method with some improvements can indeed perform better than the FMM, as has been reported by Dehnen [Dehnen02]. However, for the higher accuracy (beyond 10^{-5}) it is much slower than the FMM.

Another class of methods are the global FFT-based methods (e.g. FFTM, [Ong04]). These do not employ hierarchical data structures, and while there can be savings related to data structures because of this, some disadvantages are also obvious. One of them is related to the adaptivity. The regular FMM skips "empty" boxes, while more advanced adaptive versions (e.g. [Cheng99], [Gumerov2005]) provide substantial savings for non-uniform data. Another disadvantage is related to the use of $O(p^4)$ multipole-to-local translation operators in the FFTM (p^2 is the number of terms in the multipole expansion), which results in a total $O(p^4 N \log N)$ complexity. While the performance of the FFTM relative to the FMM should be more carefully analyzed in terms of the asymptotic constants, we note that the FMM-based algorithms of complexity $O(p^3 N)$ or even $O(Np^2 \log p)$ are available. Further, even from the viewpoint of memory utilization, these methods require more memory. Thus, while interesting and somewhat simpler to code, the method appears not competitive.

Finally, we mention some translation approaches which appear to have superior asymptotic complexity for $p \to \infty$, of $(O(p^2 \log p))$, but in practice do not appear competitive. Elliott and Board [Elliott02] proposed to use the 2D Toeplitz-Hankel structure of the translation matrices, to speed-up multiplication with them using the FFT. They reported speedups 2 and 4 for a sequential and 3 and 6 for a vector processor for $p = 8$ and $p = 16$, respectively, compared to the original $O(p^4)$ method. This shows that while formally the method is scaled as $O(p^2 \log p)$ the asymptotic constant of the method is larger than one, and, in fact, speedups for the sequential processor are even smaller than when using the above-mentioned $O(Np^3)$ method, which for $p = 8$ and 16 provides $8/3 > 2$ and $16/3 > 4$ speedups. The same situation relates to algorithms exploiting decompositions of translation operators to 1D structured matrices [Tang03], which despite lower $(O(p^2 \log p))$ asymptotic complexity have larger asymptotic constants. In practice, p is normally below 25 even for high accuracy computations, and these methods are slower than the $O(Np^3)$ methods in practice.

1.3 Organization of the paper

We describe the fast multipole method using the various translation operators in Section 2. These include the matrix-based translation (Method #0), the rotation axial-translation and rotation method (Method #1) and the plane-wave translation (Method #2). In Section 3 we provide some details of the FMM algorithm. Section 4 describes the numerical tests performed to determine the problems for which each type of algorithm may be useful. Finally section 5 provides a discussion and conclusions. Using these results one can choose the algorithm appropriate to the problem at hand.

2 Translation theory

2.1 Multipole and local expansions

Elementary solutions of the Laplace equation in three dimensions in spherical coordinates (r, θ, φ)

$$x = r \sin\theta \cos\varphi, \quad x = r \sin\theta \sin\varphi, \quad z = r \cos\theta, \tag{6}$$

can be represented as

$$R_n^m(\mathbf{r}) = \alpha_n^m r^n Y_n^m(\theta, \varphi), \quad S_n^m(\mathbf{r}) = \beta_n^m r^{-n-1} Y_n^m(\theta, \varphi), \quad n = 0, 1, ..., \quad m = -n, ..., n. \tag{7}$$

Here $R_n^m(\mathbf{r})$ are respectively the regular (local) and $S_n^m(\mathbf{r})$ the singular (far field, or multipole) spherical basis functions, α_n^m and β_n^m are normalization constants selected by convenience, and

$Y_n^m(\theta,\varphi)$ are the orthonormal spherical harmonics:

$$\begin{aligned} Y_n^m(\theta,\varphi) &= N_n^m P_n^{|m|}(\mu)e^{im\varphi}, \quad \mu=\cos\theta, \\ N_n^m &= (-1)^m\sqrt{\frac{2n+1}{4\pi}\frac{(n-|m|)!}{(n+|m|)!}}, \quad n=0,1,2,..., \quad m=-n,...,n, \end{aligned} \tag{8}$$

where $P_n^{|m|}(\mu)$ are the associated Legendre functions [Abramowitz65]. We will use the definition of the associated Legendre function $P_n^m(\mu)$ that is consistent with the value on the cut $(-1,1)$ of the hypergeometric function $P_n^m(z)$ (see Abramowitz & Stegun, [Abramowitz65]). These functions can be obtained from the Legendre polynomials $P_n(\mu)$ via the Rodrigues' formula

$$P_n^m(\mu)=(-1)^m\left(1-\mu^2\right)^{m/2}\frac{d^m}{d\mu^m}P_n(\mu), \quad P_n(\mu)=\frac{1}{2^n n!}\frac{d^n}{d\mu^n}\left(\mu^2-1\right)^n. \tag{9}$$

Our definition of spherical harmonics coincides with that of Epton & Dembart [Epton95], except for a factor $\sqrt{(2n+1)/4\pi}$, which we include to make them an orthonormal basis over the sphere. In [Cheng99] the factor $(-1)^m$ in N_n^m in Eq. (8) is omitted in what seems to be a typographical error, since further translation and conversion formulae sometimes are valid for harmonics containing this factor.

The Green's function for the Laplace equation can be written in the form

$$G(\mathbf{r},\mathbf{r}_0)=\frac{1}{4\pi|\mathbf{r}-\mathbf{r}_0|}=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\frac{R_n^{-m}(\mathbf{r}_0)S_n^m(\mathbf{r})}{\alpha_n^{-m}\beta_n^m(2n+1)}, \quad |\mathbf{r}_0|<|\mathbf{r}|. \tag{10}$$

An arbitrary harmonic function regular inside a sphere can be expanded inside this sphere into a local expansion

$$\phi(\mathbf{r})=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\phi_n^m R_n^m(\mathbf{r}). \tag{11}$$

Also an arbitrary harmonic function that decays at the infinity and is regular outside a sphere can be expanded outside this sphere as a multipole expansion

$$\phi(\mathbf{r})=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\phi_n^m S_n^m(\mathbf{r}). \tag{12}$$

The coefficients ϕ_n^m in (11) and (12) are the expansion coefficients.

Consider now a p-truncated approximation of $\phi(\mathbf{r})$ in the form

$$\phi(\mathbf{r})=\sum_{n=0}^{p-1}\sum_{m=-n}^{n}\phi_n^m F_n^m(\mathbf{r}), \quad F=S,R. \tag{13}$$

This approximation is characterized by p^2 expansion coefficients ϕ_n^m which can be stacked in a single vector, which we call the "*representing*" vector. For a general complex function the length of representation is p^2 complex numbers, or $2p^2$ real numbers. If $\phi(\mathbf{r})$ is real, then we have, e.g. for the regular expansion

$$\begin{aligned} \overline{\phi(\mathbf{r})} &= \sum_{n=0}^{p-1}\sum_{m=-n}^{n}\overline{\phi_n^m R_n^m(\mathbf{r})}=\sum_{n=0}^{p-1}\sum_{m=-n}^{n}\overline{\phi_n^m}\,\overline{\alpha_n^m}r^n\overline{Y_n^m(\theta,\varphi)} \\ &= \sum_{n=0}^{p-1}\sum_{m=-n}^{n}\overline{\phi_n^m}\,\overline{\alpha_n^m}r^n Y_n^{-m}(\theta,\varphi)=\sum_{n=0}^{p-1}\sum_{m=-n}^{n}\overline{\phi_n^{-m}}\,\overline{\alpha_n^{-m}}r^n Y_n^m(\theta,\varphi)=\sum_{n=0}^{p-1}\sum_{m=-n}^{n}\overline{\phi_n^{-m}}\frac{\overline{\alpha_n^{-m}}}{\alpha_n^m}R_n^m(\mathbf{r}). \end{aligned} \tag{14}$$

Comparing this with expansion (13) and due to the completeness of the spherical harmonic basis we have the following symmetry

$$\overline{\phi_n^{-m}\alpha_n^{-m}} = \alpha_n^m \phi_n^m. \tag{15}$$

So the real harmonic functions $\phi(\mathbf{r})$ can be represented by vectors of half the length, which store the real and imaginary parts of coefficients ϕ_n^m (e.g. if $\alpha_n^m = 1$ all ϕ_n^0 are real (p quantities) and all ϕ_n^m for $m > 0$ can be represented by $p^2 - p$ real numbers, then ϕ_n^{-m} do not require storage).

2.2 Translation operators

2.2.1 Reexpansions of elementary solutions

Elementary solutions of the Laplace equation centered at a point can be expanded in series over elementary solutions centered about some other spatial point. This can be written as the following addition theorems

$$\begin{aligned}
R_n^m(\mathbf{r}+\mathbf{t}) &= \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'} (R|R)_{n'n}^{m'm}(\mathbf{t}) R_{n'}^{m'}(\mathbf{r}), \qquad (16)\\
S_n^m(\mathbf{r}+\mathbf{t}) &= \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'} (S|R)_{n'n}^{m'm}(\mathbf{t}) R_{n'}^{m'}(\mathbf{r}), \quad |\mathbf{r}| < |\mathbf{t}|,\\
S_n^m(\mathbf{r}+\mathbf{t}) &= \sum_{n'=0}^{\infty}\sum_{m'=-n'}^{n'} (S|S)_{n'n}^{m'm}(\mathbf{t}) S_{n'}^{m'}(\mathbf{r}), \quad |\mathbf{r}| > |\mathbf{t}|,
\end{aligned}$$

where $\mathbf{t}$ is the translation vector, and $(R|R)_{n'n}^{m'm}$, $(S|R)_{n'n}^{m'm}$, and $(S|S)_{n'n}^{m'm}$ are the four index local-to-local, multipole-to-local, and multipole-to-multipole reexpansion coefficients. Explicit expressions for these coefficients can be found elsewhere (see e.g. [Epton95, Cheng99]).

We define the following complex normalization factors

$$\begin{aligned}
\alpha_{(1)n}^m &= (-1)^n i^{-|m|}\sqrt{\frac{4\pi}{(2n+1)(n-m)!(n+m)!}}, \quad \beta_{(1)n}^m = i^{|m|}\sqrt{\frac{4\pi(n-m)!(n+m)!}{2n+1}}, \qquad (17)\\
n &= 0, 1, ..., \quad m = -n, ..., n.
\end{aligned}$$

If we set then $\alpha_n^m = \alpha_{(1)n}^m$ and $\beta_n^m = \beta_{(1)n}^m$ in the definitions of the basis functions (7) then the reexpansion coefficients take a particularly simple form (see [Epton95]):

$$\begin{aligned}
(R|R)_{n'n}^{m'm}(\mathbf{t}) &= R_{n-n'}^{m-m'}(\mathbf{t}), \quad |m'| \leqslant n', \qquad (18)\\
(S|R)_{n'n}^{m'm}(\mathbf{t}) &= S_{n+n'}^{m-m'}(\mathbf{t}), \quad |m'| \leqslant n', \quad |m| \leqslant n,\\
(S|S)_{n'n}^{m'm}(\mathbf{t}) &= R_{n'-n}^{m-m'}(\mathbf{t}), \quad |m| \leqslant n.
\end{aligned}$$

In the other commonly used case the normalization constants are selected as $\alpha_n^m = 1$ and $\beta_n^m = 1$. In this case we have

$$\begin{aligned}
(R|R)_{n'n}^{m'm} &= \frac{\alpha_{(1)n'}^{m'}\alpha_{(1)n-n'}^{m-m'}}{\alpha_{(1)n}^m} R_{n-n'}^{m-m'}(\mathbf{t}), \qquad (19)\\
(S|R)_{n'n}^{m'm} &= \frac{\alpha_{(1)n'}^{m'}\beta_{(1)n+n'}^{m-m'}}{\beta_{(1)n}^m} S_{n+n'}^{m-m'}(\mathbf{t}),\\
(S|S)_{n'n}^{m'm} &= \frac{\beta_{(1)n'}^{m'}\alpha_{(1)n'-n}^{m-m'}}{\beta_{(1)n}^m} R_{n'-n}^{m-m'}(\mathbf{t}),
\end{aligned}$$

where $\alpha^m_{(1)n}$ and $\beta^m_{(1)n}$ are given by Eq. (17).

2.2.2 Translation using truncated reexpansion matrices (Method #0)

Arranging the reexpansion coefficients to form a truncated reexpansion matrices yields a straightforward translation method (here and below we refer to this as Method #0). Let $\phi(\mathbf{r})$ be an arbitrary scalar function, $\phi : \Omega(\mathbf{r}) \to \mathbb{C}$, where $\Omega(\mathbf{r}) \subset \mathbb{R}^3$. For a given vector $\mathbf{t} \in \mathbb{R}^3$ We define a new function $\widehat{\phi} : \widehat{\Omega}(\mathbf{r}) \to \mathbb{C}$, $\widehat{\Omega}(\mathbf{r}) \subset \mathbb{R}^3$ such that in $\widehat{\Omega}(\mathbf{r}) = \Omega(\mathbf{r}+\mathbf{t})$ the values of $\widehat{\phi}(\mathbf{r})$ coincide with the values of $\phi(\mathbf{r}+\mathbf{t})$ and we treat $\widehat{\phi}(\mathbf{r})$ as a result of the action of the translation operator $\mathcal{T}(\mathbf{t})$ on $\phi(\mathbf{r})$:

$$\widehat{\phi} = \mathcal{T}(\mathbf{t})[\phi], \quad \widehat{\phi}(\mathbf{r}) = \phi(\mathbf{r}+\mathbf{t}), \quad \mathbf{r} \in \widehat{\Omega}(\mathbf{r}) \subset \mathbb{R}^3. \tag{20}$$

The function can be represented by the (infinite) vector of coefficients multiplying their associated basis functions. If $\phi(\mathbf{r})$ be a regular solution of the Laplace equation inside a sphere Ω_a of radius a, that includes the origin, it can be represented in the form

$$\phi(\mathbf{r}) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \phi^m_n R^m_n(\mathbf{r}), \tag{21}$$

where ϕ^m_n are the expansion coefficients over the basis $\{R^m_n(\mathbf{r})\}$. Similarly a solution of Laplace's equation $\phi(\mathbf{r})$ regular outside a sphere Ω_a can be expanded over the basis $\{S^m_n(\mathbf{r})\}$. The translated function $\widehat{\phi}(\mathbf{r})$ can be also be expanded over the bases $\{R^m_n(\mathbf{r})\}$ or $\{S^m_n(\mathbf{r})\}$ with expansion coefficients $\widehat{\phi}^m_n$. Due to linearity of the translation operator the sets $\left\{\widehat{\phi}^m_n\right\}$ and $\{\phi^m_n\}$ will be related by a translation operator (an infinite matrix), which represents the translation operator in the respective bases.

If $\widehat{\phi}$ is expanded over the same basis as ϕ, Eq. (21), in $\Omega_a \subset \widehat{\Omega}(\mathbf{r}) \cap \Omega(\mathbf{r})$:

$$\widehat{\phi}(\mathbf{r}) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \widehat{\phi}^m_n R^m_n(\mathbf{r}). \tag{22}$$

Then we have

$$\begin{aligned} \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \widehat{\phi}^m_n R^m_n(\mathbf{r}) &= \widehat{\phi}(\mathbf{r}) = \phi(\mathbf{r}+\mathbf{t}) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} \phi^{m'}_{n'} R^{m'}_{n'}(\mathbf{r}+\mathbf{t}) \\ &= \sum_{n=0}^{\infty} \sum_{m=-n}^{n} \left[\sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (R|R)^{mm'}_{nn'}(\mathbf{t}) \phi^{m'}_{n'} \right] R^m_n(\mathbf{r}), \end{aligned} \tag{23}$$

which shows that

$$\widehat{\phi}^m_n = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} (R|R)^{mm'}_{nn'}(\mathbf{t}) \phi^{m'}_{n'}, \tag{24}$$

assuming that all the series converge absolutely and uniformly.

It is not difficult to prove that for any ϵ we can find some p such that

$$\left| \widehat{\phi}(\mathbf{r}) - \sum_{n=0}^{p-1} \sum_{m=-n}^{n} \widehat{\phi}^{(p)m}_n R^m_n(\mathbf{r}) \right| < \epsilon, \quad \widehat{\phi}^{(p)m}_n = \sum_{n'=0}^{p-1} \sum_{m'=-n'}^{n'} (R|R)^{mm'}_{nn'}(\mathbf{t}) \phi^{m'}_{n'}. \tag{25}$$

We call p the truncation number. The number of coefficients $\widehat{\phi}_n^{(p)m}$ is p^2 and to perform the local-to-local translation we multiply a dense matrix $\left\{(R|R)_{nn'}^{mm'}(\mathbf{t})\right\}$ by the input representing vector $\left\{\phi_{n'}^{m'}\right\}$. A straightforward multiplication by the translation operation will take $O(p^4)$ multiplications/additions, even though the coefficients $(R|R)_{nn'}^{mm'}$ can be stacked into upper triangular matrix.

Similar consideration can be given to multipole-to-local and multipole-to-multipole translation operators, which are represented by matrices $(S|R)_{n'n}^{m'm}(\mathbf{t})$ and $(S|S)_{n'n}^{m'm}(\mathbf{t})$, respectively. For details we refer to [Greengard87], [Epton95], [Greengard97].

The translation matrices have Toeplitz/Hankel structure – and this fact that can be potentially exploited for faster matrix-vector multiplication [Elliott02] to reduce formally the procedure to $O(p^2 \log p)$ operations.However the use of the fast Fourier transforms in the translation process can be justified only at sufficiently large p due to large asymptotic constants in the $O(p^2 \log p)$ complexity dependence. Normally, such values of p are not encountered in the solution of the Laplace equation, where sufficient accuracy can be achieved at smaller p.

Finally we note that in the FMM we do not translate the function, but change the center of expansion. For example, under local-to-local translation from center $\mathbf{r}_1$ to center $\mathbf{r}_2$ we mean representation of the same function in the regular bases centered at these point respectively. Since for representations of the same function we have

$$\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\phi_n^m R_n^m(\mathbf{r}-\mathbf{r}_1)=\sum_{n=0}^{\infty}\sum_{m=-n}^{n}\widehat{\phi}_n^m R_n^m(\mathbf{r}-\mathbf{r}_2), \tag{26}$$

it is not difficult to see that the expansion coefficients are related by Eq. (24), where the translation vector is $\mathbf{t}=\mathbf{r}_2-\mathbf{r}_1$.The same relates to the multipole-to-local and multipole-to-multipole translations, where we use the $S|R$ and $S|S$ matrices instead of the $R|R$ translation matrix. Figure 1 provides illustration of the different types of translation operators used in the FMM.

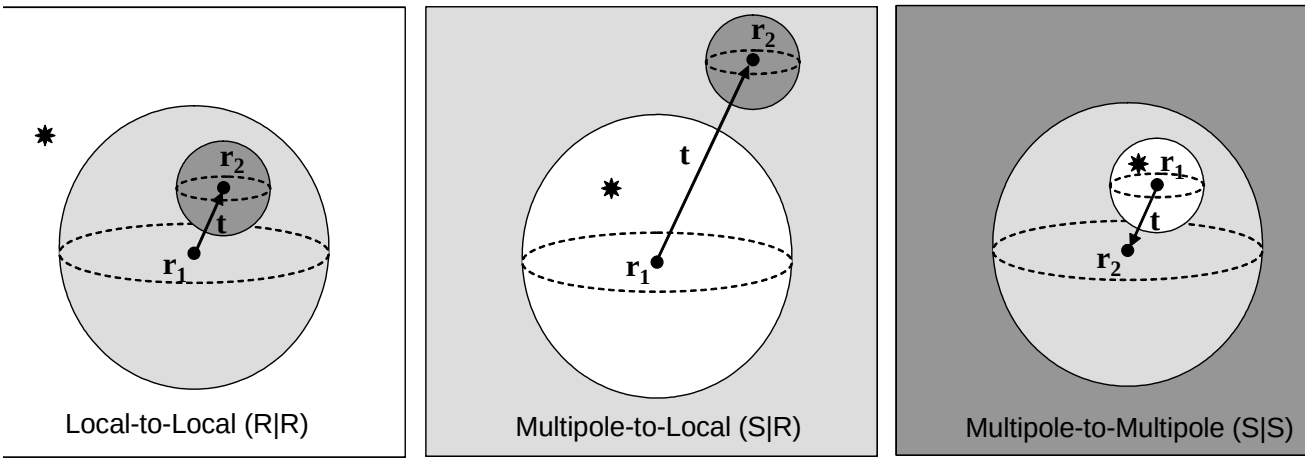

Figure 1: Illustration of local-to-local (R|R), multipole-to-local (S|R), and multipole-to-multipole (S|S) translations from expansion center $\mathbf{r}_1$ to expansion center $\mathbf{r}_2$ (translation vector $\mathbf{t}=\mathbf{r}_2-\mathbf{r}_1$). The star shows location of a source (singular point). Expansion about center $\mathbf{r}_1$ is valid inside the lighter and darker gray area, while expansion about center $\mathbf{r}_2$ is valid only in the darker gray area (subdomain of the lighter gray area). The source (sources) can be located in the white domains.

2.3 "Point-and-shoot" translation method (Method #1)

The "point-and-shoot" translation method (using the colorful terminology suggested by Rokhlin), which we refer as Method #1, is based on decomposition of the translation operator to the product of rotation and coaxial translation operators. This method was described first in Ref. [White96].

2.3.1 Coaxial Translations

The coaxial translation is the translation along the z-coordinate axis, i.e. this is the case when the translation vector $\mathbf{t} = t\mathbf{i}_z$, where $\mathbf{i}_z$ is the basis unit vector for the z-axis. The peculiarity of the coaxial translation is that it does not change the order m of the translated coefficients, and so translation can be performed for each order independently. For example, Eq. (24) for the coaxial local-to-local translation will be reduced to

$$\widehat{\phi}_n^m = \sum_{n'=|m|}^{\infty} (R|R)_{nn'}^m (t)\phi_{n'}^m, \quad m = 0, \pm 1, ..., \quad n = |m|, |m|+1, ... \tag{27}$$

The three index coaxial reexpansion coefficients $(F|E)_{nn'}^m$ ($F, E = S, R$; $m = 0, \pm 1, \pm 2, ...,$ $n, n' = |m|, |m|+1, ...$) are functions of the translation distance t only and can be expressed via the general reexpansion coefficients as

$$(F|E)_{nn'}^m (t) = (F|E)_{nn'}^{mm} (t\mathbf{i}_z), \quad F, E = S, R; \quad t \geqslant 0. \tag{28}$$

Using Eq. (18) we have for normalized basis functions (7) with $\alpha_n^m = \alpha_{(1)n}^m$ and $\beta_n^m = \beta_{(1)n}^m$:

$$\begin{aligned} (R|R)_{nn'}^m (t) &= r_{n'-n}(t), \quad n' \geqslant |m|, \\ (S|R)_{nn'}^m (t) &= s_{n+n'}(t), \quad n, n' \geqslant |m|, \\ (S|S)_{nn'}^m (t) &= r_{n-n'}(t), \quad n \geqslant |m|, \end{aligned} \tag{29}$$

where the functions $r_n(t)$ and $s_n(t)$ are

$$r_n(t) = \frac{(-t)^n}{n!}, \quad s_n(t) = \frac{n!}{t^{n+1}} \quad n = 0, 1, ..., \quad t \geqslant 0, \tag{30}$$

and zero for $n < 0$. This show that for given m matrices $\{(R|R)_{nn'}^m (t)\}$ are upper triangular, $\{(S|S)_{nn'}^m (t)\}$ are lower triangular, and $\{(S|R)_{nn'}^m (t)\}$ is a fully populated matrix. The latter matrix is symmetric, while $\{(S|S)_{nn'}^m (t)\} = \{(R|R)_{n'n}^m (t)\}$, i.e. these matrices are transposes of each other. It is also important to note that the coaxial translation matrices are real.

It is not difficult then to write expressions for coaxial translation coefficients in the basis of functions normalized with $\alpha_n^m = 1$ and $\beta_n^m = 1$ (see Eqs. (17), (19), and (29)). In this basis all matrices remain real, but lose the symmetry properties, mentioned above.

For p-truncated function representations by its expansion coefficients, the number of operations required to perform a single coaxial translation with dense matrix (multipole-to-local) is

$$N_{Coax}(p) = \sum_{m=-(p-1)}^{p-1} \left(\sum_{n=|m|}^{p-1} 1 \right)^2 = \sum_{m=-(p-1)}^{p-1} (p - |m|)^2 = \frac{1}{3} p \left(2p^2 + 1\right). \tag{31}$$

This number is obtained assuming that 1 operation is spent for multiplication/addition (assuming that the translation matrices can be precomputed and stored). Note that this estimate is valid for real ϕ_n^m. Generally, these coefficients are complex, and so for multipole-to-local translation one

should spent $2N_{Coax}(p)$ real multiplications/additions. In case when ϕ_n^m represent real function $\phi(\mathbf{r})$ the count of operations goes back to $N_{Coax}(p)$ due to the symmetry of expansion coefficients (e.g. for functions normalized with $\alpha_n^m = \beta_n^m = 1$ we have $\phi_n^{-m} = \overline{\phi_n^m}$ and with $\alpha_n^m = \alpha_{(1)n}^m$, $\beta_n^m = \beta_{(1)n}^m$ symmetry $\phi_n^{-m} = (-1)^m \overline{\phi_n^m}$ holds). Further we will count only operations required for translations of real functions, since the complex functions can be computed by independent transforms of the real and imaginary parts (so the number of operations simply doubles).

Note that for local-to-local and multipole-to-multipole coaxial translations we have the following estimate

$$\begin{aligned} N_{Coax}^{(R|R)}(p) &= N_{Coax}^{(S|S)}(p) = \sum_{m=-(p-1)}^{p-1} \sum_{n=|m|}^{p-1} \sum_{n'=|m|}^{n} = \frac{1}{2} \sum_{m=-(p-1)}^{p-1} (p-|m|)(p-|m|+1) \qquad (32) \\ &= \frac{1}{2}\left[N_{Coax}(p) + p^2\right]. \end{aligned}$$

This is because the corresponding matrices are upper or lower triangular.

2.3.2 Rotations

To perform translation with an arbitrary vector $\mathbf{t}$ using the computationally cheap coaxial translation operators, we first must rotate the original reference frame to align the z-axis of the rotated reference frame with $\mathbf{t}$, translate and then perform an inverse rotation.

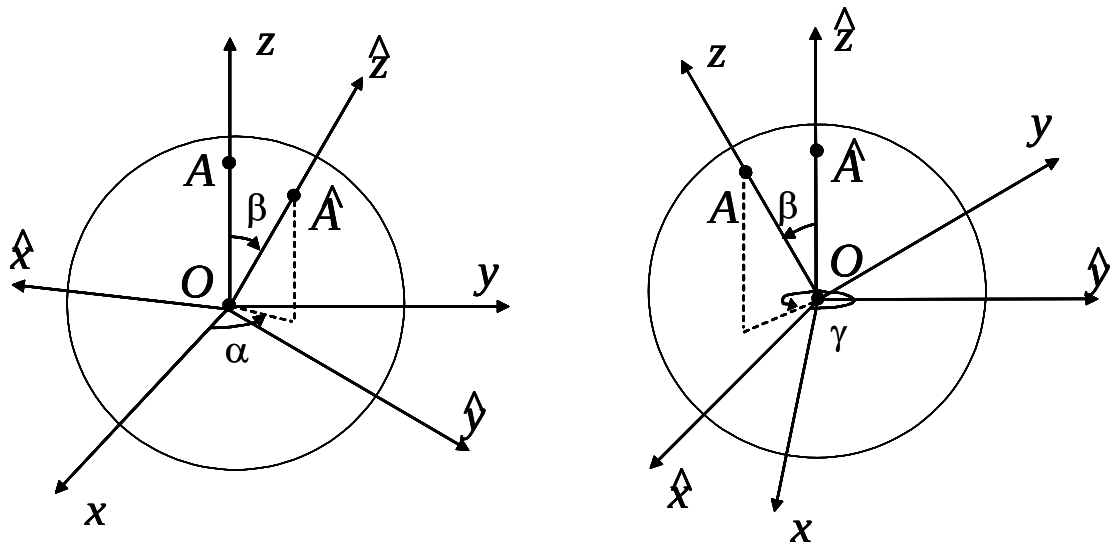

Figure 2: The figure on the left shows the transformed axes $(\hat{x}, \hat{y}, \hat{z})$ in the original reference frame (x, y, z). The spherical polar coordinates of the point $\hat{A}$ lying on the $\hat{z}$ axis on the unit sphere are (β, α). The figure on the right shows the original axes (x, y, z) in the transformed reference frame $(\hat{x}, \hat{y}, \hat{z})$. The coordinates of the point A lying on the z axis on the unit sphere are (β, γ). The points O, A, and $\hat{A}$ are the same in both figures. All rotation matrices can be derived in terms of these three angles α, β, γ.

An arbitrary rotation in three dimensions can be characterized by three Euler angles, or angles α, β, and γ that are simply related to them. For the forward rotation, when (θ, φ) are the spherical

polar angles of the rotated z-axis in the original reference frame, then $\beta = \theta$, $\alpha = \varphi$; for the inverse rotation with $\left(\widehat{\theta}, \widehat{\varphi}\right)$ the spherical polar angles of the original z-axis in the rotated reference frame, $\beta = \widehat{\theta}$, $\gamma = \widehat{\varphi}$ (see Fig. 2). An important property of the spherical harmonics is that on rotation their degree n does not change, i.e.

$$Y_n^m(\theta,\varphi) = \sum_{m'=-n}^{n} T_n^{m'm}(\alpha,\beta,\gamma)\, Y_n^{m'}\left(\widehat{\theta},\widehat{\varphi}\right), \quad n = 0,1,2,..., \quad m = -n,...,n, \tag{33}$$

where (θ,φ) and $\left(\widehat{\theta},\widehat{\varphi}\right)$ are spherical polar angles of the same point on the unit sphere in the original and the rotated reference frames, and $T_n^{m'm}(\alpha,\beta,\gamma)$ are the rotation coefficients.

Rotation transform for solution of the Laplace equation factorized over the regular spherical basis functions can be performed as.

$$\begin{aligned}\phi(\mathbf{r}) &= \sum_{n=0}^{\infty}\sum_{m=-n}^{n} \phi_n^m R_n^m(\mathbf{r}) = \sum_{n=0}^{\infty} r^n \sum_{m=-n}^{n} \phi_n^m \alpha_n^m Y_n^m(\theta,\varphi) \\ &= \sum_{n=0}^{\infty}\sum_{m'=-n}^{n}\left[\sum_{m=-n}^{n} T_n^{m'm}(\alpha,\beta,\gamma)\,\alpha_n^m\phi_n^m\right] r^n Y_n^{m'}\left(\widehat{\theta},\widehat{\varphi}\right) = \sum_{n=0}^{\infty}\sum_{m=-n}^{n} \widehat{\phi}_n^m R_n^m(\widehat{\mathbf{r}}),\end{aligned} \tag{34}$$

where $\mathbf{r}$ and $\widehat{\mathbf{r}}$ are coordinates of the same field point in the original and rotated frames, while ϕ_n^m and $\widehat{\phi}_n^m$ are the respective expansion coefficients related as

$$\widehat{\phi}_n^m = \sum_{m'=-n}^{n} \frac{T_n^{mm'}(\alpha,\beta,\gamma)\,\alpha_n^{m'}}{\alpha_n^m}\phi_n^{m'}. \tag{35}$$

The same holds for the multipole expansions where in Eq. (35) one should replace α_n^m and $\alpha_n^{m'}$ normalization constants with β_n^m and $\beta_n^{m'}$, respectively. In case $\alpha_n^m = \beta_n^m$ the rotation coefficients for the regular and singular basis functions are the same.

Rotation coefficients $T_n^{m'm}(\alpha,\beta,\gamma)$ can be decomposed as

$$T_n^{m'm}(\alpha,\beta,\gamma) = e^{im\alpha} e^{-im'\gamma} H_n^{m'm}(\beta), \tag{36}$$

where $\left\{H_n^{m'm}(\beta)\right\}$ is a dense real symmetric matrix. Its entries can be computed using an analytical expression, or by a fast recursive procedure (see [Gumerov2005]), which starts with the initial value

$$H_n^{m'0}(\beta) = (-1)^{m'}\sqrt{\frac{(n-|m'|)!}{(n+|m'|)!}} P_n^{|m'|}(\cos\beta), \quad n = 0,1,..., \quad m' = -n,...,n, \tag{37}$$

and further propagates for positive m:

$$H_{n-1}^{m',m+1} = \frac{1}{b_n^m}\left\{\frac{1}{2}\left[b_n^{-m'-1}(1-\cos\beta)\,H_n^{m'+1,m} - b_n^{m'-1}(1+\cos\beta)\,H_n^{m'-1,m}\right] - a_{n-1}^{m'}\sin\beta H_n^{m'm}\right\}, \tag{38}$$

$$n = 2,3,..., \quad m' = -n+1,...,n-1, \quad m = 0,...,n-2,$$

where $a_n^m = b_n^m = 0$ for $n < |m|$, and

$$a_n^m = a_n^{-m} = \sqrt{\frac{(n+1+m)(n+1-m)}{(2n+1)(2n+3)}}, \quad \text{for } n \geqslant |m|, \tag{39}$$

$$b_n^m = \begin{cases} \sqrt{\frac{(n-m-1)(n-m)}{(2n-1)(2n+1)}}, & 0 \leqslant m \leqslant n, \\ -\sqrt{\frac{(n-m-1)(n-m)}{(2n-1)(2n+1)}}, & -n \leqslant m < 0. \end{cases}.$$

For negative m coefficients $H_n^{m'm}(\beta)$ can be found using symmetry $H_n^{-m',-m}(\beta) = H_n^{m'm}(\beta)$.

We note that inverse rotation can be performed using the matrix $\left\{\left(T^{-1}\right)_n^{m'm}(\alpha,\beta,\gamma)\right\}$, which is the complex conjugate transposed of $\left\{T_n^{m'm}(\alpha,\beta,\gamma)\right\}$ and can be simplified using Eq. (36):

$$\left(T^{-1}\right)_n^{m'm}(\alpha,\beta,\gamma) = \overline{T_n^{mm'}(\alpha,\beta,\gamma)} = e^{-im'\alpha}e^{im\gamma}H_n^{mm'}(\beta) = e^{-im'\alpha}e^{im\gamma}H_n^{m'm}(\beta) = T_n^{m'm}(\gamma,\beta,\alpha). \tag{40}$$

In the "point-and-shoot" method the angle γ can be selected arbitrarily, since the direction of the translation vector $\mathbf{t}$ is characterized only by the two angles, α and β. For example, one could simply set $\gamma = 0$. We found however, that setting $\gamma = \alpha$ can be computationally cheaper, since in this case the forward and inverse translation operators coincide, $\left\{\left(T^{-1}\right)_n^{m'm}(\alpha,\beta,\alpha)\right\} = \left\{T_n^{m'm}(\alpha,\beta,\alpha)\right\}$ (for the normalization $\alpha_n^m = \beta_n^m = 1$).

In any case, the entries of the rotation matrix can be precomputed and stored. The number of operations required for one rotation of vector of size p^2 then can be counted as

$$N_{Rot}(p) = \sum_{n=0}^{p-1}\left(\sum_{m=-n}^{n} 1\right)^2 = \sum_{n=0}^{p-1}(2n+1)^2 = \frac{1}{3}p\left(4p^2-1\right). \tag{41}$$

As before we assumed here that one operation is required for multiplication/addition (precomputed rotation matrices). This is true for rotation of coefficients of real functions, due to the above mentioned symmetry, and this number should be multiplied by 2 for general complex coefficients. In the latter case we note that decomposition (36) with $\gamma = 0$ can be more efficient, since complex multiplications should be performed only for diagonal matrix with entries $e^{im\alpha}$ ($O(p^2)$ operations) while dense matrix multiplication involves real matrix $\left\{H_n^{m'm}(\beta)\right\}$.

The operation count (31) and (41) brings the total complexity of the multipole-to-local translation of coefficients of real functions for the Method #1 to

$$N_1^{(S|R)}(p) = N_{Rot}(p) + N_{coax}(p) + N_{Rot}(p) = \frac{1}{3}p\left(2p^2+1\right) + \frac{2}{3}p\left(4p^2-1\right) \approx \frac{10}{3}p^3. \tag{42}$$

Comparing this with the complexity of the same operation using straightforward multiplication of a dense matrix (Method #0), which we refer as

$$N_0^{(S|R)}(p) = p^4, \tag{43}$$

we can see that for $p \geqslant 4$ Method #1 should be theoretically faster than Method #0.

We also note that for local-to-local and multipole-to-multipole translations, the number of operations using Method #1 can be evaluated as (see Eq. (32)):

$$N_1^{(S|S)}(p) = N_{Rot}(p) + N_{coax}^{(S|S)}(p) + N_{Rot}(p) = \frac{1}{6}p\left(2p^2+1\right) + \frac{1}{2}p^2 + \frac{2}{3}p\left(4p^2-1\right) \approx 3p^3. \tag{44}$$

2.4 Exponential expansions

One of the most expensive operation in the FMM is the multipole-to-local translation. While the cost of a single $S|R$-translation operation is comparable with the cost of the $R|R$- and $S|S$-operations, the number of multipole-to-local translations is much larger (approximately hundred times). To reduce the cost of this operation Greengard and Rokhlin [Greengard97] (see also Cheng *et*

al. [Cheng99]) proposed to use the exponential expansions (also referred as "plane wave" expansions in analogy with similar expansions introduced for the wave equation; physically, this is not correct, since the Laplace equation is an elliptic equation). The plane wave expansions are based on the following representation of the Green's function (10) in the domain $z > z_0$:

$$\begin{aligned} G(\mathbf{r},\mathbf{r}_0) &= \frac{1}{8\pi^2}\int_0^\infty e^{-\lambda(z-z_0)}\int_0^{2\pi} e^{i\lambda[(x-x_0)\cos\alpha+(y-y_0)\sin\alpha]}d\alpha d\lambda, \\ \mathbf{r} &= (x,y,z), \quad \mathbf{r}_0=(x_0,y_0,z_0), \quad z>z_0. \end{aligned} \tag{45}$$

In Refs. [Greengard97, Cheng99] the following quadrature for the integrals is proposed when

$$a \leqslant z-z_0 \leqslant b, \sqrt{(x-x_0)^2+(y-y_0)^2} \leqslant c, \tag{46}$$

where a, b and c are some numbers (for the 3-D FMM with the standard 1-neighborhoods [Gumerov2003], the approximation should be valid for $a=1$, $b=4$, $c=4\sqrt{2}$):

$$G(\mathbf{r},\mathbf{r}_0) = \frac{1}{4\pi}\sum_{k=1}^{S(\epsilon)}\frac{w_k}{M_k}e^{-\lambda_k(z-z_0)}\sum_{j=1}^{M_k}e^{i\lambda_k\left[(x-x_0)\cos\alpha_{jk}+(y-y_0)\sin\alpha_{jk}\right]}, \tag{47}$$

where λ_k and $\alpha_{jk}=2\pi j/M_k$ $(j=1,...,M_k)$ are the quadrature nodes, w_k are the weights, M_k is the number of nodes in the kth subspace where the discrete Fourier transform with respect to α is performed, and $S(\epsilon)$ is some number dependent on the accuracy of approximation, ϵ. As this number varies the quadrature nodes and weights as well as M_k vary also. For particular cases with $S(\epsilon)=8,17,26$, which we refer as the order of quadrature, the nodes and weights can be found in Refs. [Greengard97, Cheng99].

This quadrature generates a more general representation valid not only for the Green function, but for an arbitrary multipole expansion of the Laplace equation, that can be viewed as a sum of sources located inside some domain. So if a harmonic function $\phi(\mathbf{r})$ is regular in Ω and has all singularities inside Ω_0, such that and for any $\mathbf{r}_0\in\Omega_0$ and $\mathbf{r}\in\Omega$ condition (46) holds, then $\phi(\mathbf{r})$ can approximated in Ω with the specified accuracy ϵ with the following expansion:

$$\phi(\mathbf{r}) = \sum_{k=1}^{S(\epsilon)}\sum_{j=1}^{M_k} W_{kj}e^{-\lambda_k z}e^{i\lambda_k\left(x\cos\alpha_{jk}+y\sin\alpha_{jk}\right)}, \tag{48}$$

where W_{kj} are the expansion coefficients.

The exponential expansion, therefore, provides an alternative to the spherical multipole and local expansions, and is generated by expansion of a harmonic function in cylindrical coordinates (in a bounded cylindrical domain). The length of a representation using this expansion is

$$S_{\text{exp}} = \sum_{k=1}^{S(\epsilon)} M_k \tag{49}$$

complex numbers or $2S_{\text{exp}}$ real numbers. This can be reduced by half for real functions due to the symmetry

$$W_{kj} = W_{k,j+M_k/2}, \quad j=1,...,M_k/2. \tag{50}$$

Indeed, we have

$$\alpha_{j+M_k/2,k} = \frac{2\pi\left(j+M_k/2\right)}{M_k} = \pi+\alpha_{jk}, \quad j=1,...,M_k/2. \tag{51}$$

So

$$e^{i\lambda_k\left(x\cos\alpha_{j+M_k/2,k}+y\sin\alpha_{j+M_k/2,k}\right)}=e^{-i\lambda_k\left(x\cos\alpha_{jk}+y\sin\alpha_{jk}\right)}=\overline{e^{i\lambda_k\left(x\cos\alpha_{jk}+y\sin\alpha_{jk}\right)}}, \tag{52}$$

and comparing expansions (48) for $\phi(\mathbf{r})$ and $\overline{\phi(\mathbf{r})}$, that should be the same, we obtain Eq. (50).

Finally we note that $S(\epsilon)$ and $S_{\rm exp}$ are related to the truncation number p, which provides the same error ϵ as

$$S(\epsilon)=\kappa p,\quad S_{\rm exp}=\sigma p^2, \tag{53}$$

where κ and σ are some constants, which can be found from numerical tests.

2.5 Exponential translation and conversion (Method #2)

The exponential expansion diagonalizes the translation operator, since the exponents are its eigenfunctions. Based on Eq. (49) we have

$$\begin{aligned}\phi(\mathbf{r}+\mathbf{t}) &= \sum_{k=1}^{S(\epsilon)}\sum_{j=1}^{M_k} W_{kj}e^{-\lambda_k t_z}e^{i\lambda_k\left(t_x\cos\alpha_{jk}+t_y\sin\alpha_{jk}\right)}e^{-\lambda_k z}e^{i\lambda_k\left(x\cos\alpha_{jk}+y\sin\alpha_{jk}\right)} \\ &= \sum_{k=1}^{S(\epsilon)}\sum_{j=1}^{M_k}\widehat{W}_{kj}e^{-\lambda_k z}e^{i\lambda_k\left(x\cos\alpha_{jk}+y\sin\alpha_{jk}\right)}=\widehat{\phi}(\mathbf{r}),\end{aligned} \tag{54}$$

where $\widehat{W}_{kj}$ are the coefficients of the translated function $\widehat{\phi}(\mathbf{r})$:

$$\widehat{W}_{kj}=E_{kj}W_{kj},\quad E_{kj}=e^{-\lambda_k t_z}e^{i\lambda_k\left(t_x\cos\alpha_{jk}+t_y\sin\alpha_{jk}\right)},\quad \mathbf{t}=(t_x,t_y,t_z). \tag{55}$$

If the exponential translation coefficients E_{kj} are precomputed and stored, then translation requires $N_{rep}^{\rm exp}$ complex-by-complex multiplications for translation of a complex function and $N_{rep}^{\rm exp}$ complex-by-real multiplications for translation of real function. So the number of real multiplication operations for translation of coefficients of real function is

$$N^{(E|E)}(S_{\rm exp})=2S_{\rm exp}=2\sigma p^2, \tag{56}$$

and double this for the complex case.

In the version of the FMM in [Greengard97, Cheng99] the exponential translation is applied only to reduce the cost of the multipole-to-local translations, while multipole-to-multipole, and local-to-local translations are performed with a traditional scheme. This method, therefore, also requires multipole-to-exponential and exponential-to-local conversions.

Note that Eq. (54) is given in the form that follows from Eq. (48). Since the FMM employs a hierarchical data structure with different size boxes at different levels, the nodes and weights of the quadrature should be scaled to appropriate box size d. Namely, the nodes λ_k and weights w_k in the quadrature should be replaced by λ_k/d and w_k/d as λ_k and w_k are obtained for a unit box (indeed multiplication of $\mathbf{r}$ by d reduces the value of the Green function d times). According to the [Greengard97, Cheng99] and our normalization of the spherical harmonics the multipole-to-exponential, or $S|E$, transform can be performed as

$$W_{kj}=\frac{w_k}{M_k d}\sum_{m=-(p-1)}^{p-1}e^{im\alpha_{jk}}\sum_{n=|m|}^{p-1}\frac{1}{\beta_n^m\beta_{(1)n}^m}\left(\frac{\lambda_k}{d}\right)^n\phi_n^m,\quad k=1,..,s(\epsilon),\quad j=1,...,M_k, \tag{57}$$

where $\beta^m_{(1)n}$ is provided by Eq. (17) and β^m_n is an arbitrary normalization factor for the singular basis functions (7).

Being thought of as a linear operation with matrix of size $N^{\exp}_{rep} \times p^2$ the above conversion requires $O\left(N^{\exp}_{rep} p^2\right)$ operation. However, Eq. (57) provides a subspace decomposition of the conversion operation (similar to coaxial translation), since first the inner sum can be computed for all m and k and then the outer sum can be computed for all k and j. For precomputed and stored coefficients computation of the inner sum requires $s\left(\epsilon\right)\left(p - |m|\right)$ operations for the mth subspace and all k. Note that if coefficients β^m_n selected to be equal $\beta^m_{(1)n}$ then coefficients in the inner sum are real (if $\beta^m_n = 1$ factor $i^{|m|}$ from $\beta^m_{(1)n}$ can be taken out of the inner summation). This means that the number of real multiplication for all subspaces (taking into account the symmetry of ϕ^m_n for real functions) is

$$N^{(S|E)}_{(inner)} = S\left(\epsilon\right) \sum_{m=-(p-1)}^{p-1} \left(p - |m|\right) = S\left(\epsilon\right) \sum_{m=-(p-1)}^{p-1} \left(p - |m|\right) = S\left(\epsilon\right) p^2. \tag{58}$$

Computation of the outer sum requires then M_k complex multiplications per each element (m, k). This results in the following count of real multiplications:

$$N^{(S|E)}_{(outer)} = S_{\exp} \sum_{m=-(p-1)}^{p-1} 2 = 2\left(2p - 1\right) S_{\exp}. \tag{59}$$

This count includes symmetry of the expansion coefficient (50). Therefore, the actual complexity of the $(S|E)$ operation is

$$N^{(S|E)} = N^{(S|E)}_{(inner)} + N^{(S|E)}_{(outer)} = S\left(\epsilon\right) p^2 + 2\left(2p - 1\right) S_{\exp} \approx \left(\kappa + 4\sigma\right) p^3. \tag{60}$$

Note that this number is referred in [Cheng99] as $O(p^3)$.

The second conversion operation is conversion of the exponential expansion to local expansion, or $E|R$ operation. This can be performed using the following formula (we make some corrections to what appear to be typographical errors in [Cheng99] and use our definition of the spherical harmonics):

$$\phi^m_n = \alpha^m_n \alpha^m_{(1)n} \sum_{k=1}^{S(\epsilon)} \left(\frac{\lambda_k}{d}\right)^n \sum_{j=1}^{M_k} W_{kj} e^{-im\alpha_{jk}}, \tag{61}$$

where $\alpha^m_{(1)n}$ is provided by Eq. (17) and α^m_n is an arbitrary normalization constant for the regular basis functions (7). It is not difficult to see that the operation count for the $E|R$ conversion is exactly the same as for the $S|E$ conversion (k inverse discrete Fourier transforms instead of the forward ones, and one multiplication by real matrix for the mth subspace). So we have for representations of real functions the number of real multiplications

$$N^{(E|R)} = N^{(S|E)} = S\left(\epsilon\right) p^2 + 2\left(2p - 1\right) S_{\exp} \approx \left(\kappa + 4\sigma\right) p^3. \tag{62}$$

The advantage of the exponential expansions is that algorithmically expensive conversion operations can be performed far fewer times than the translation. Details of this can be found in [Cheng99] (see also the next section) and we refer the translation method using the exponential expansions as Method #2.

3 Algorithms

We will not present details of the basic FMM algorithm, which are well described in the original papers of Greengard, Rokhlin, and others [Greengard87], [Greengard88]. We describe some flow charts and particulars of our implementation. The basic algorithm can be designed for low memory consumption (e.g. see [Gumerov2003]), though such an approach is substantially slower than the algorithm storing data structures and using precomputed translation data, that was used in the current test.

The algorithm consists of two main parts: the preset step, which includes setting the data structure (building and storage of the neighbor lists, etc.) and precomputation and storage of all translation data. The data structure is generated using the bit interleaving technique described in [Gumerov2003], which enables spatial ordering, sorting, and bookmarking. The algorithm is designed for two independent data sets (N arbitrary located sources and M arbitrary evaluation points), while for the current tests we usually used the same source and evaluation sets of length N, which is also called the problem size. For a problem size N, the cost of building the data structure based on spatial ordering is $O(N \log N)$, where the asymptotic constant is much smaller than the constants in the $O(N)$ asymptotics of the main algorithm. The number of levels could be arbitrarily set by the user or found automatically based on the clustering parameter (the maximum number of sources in the smallest box) for optimization of computations of problems of different size.

For efficient storage and retrieval of translation data we introduced a translation index, which shows the direction of the translations (8 indices for each level for the $S|S$ and $R|R$ translations and 343 indices for the $S|R$ translations for each level). For the use of Method #2, which requires also specification of the 6 directions of the "plane wave" propagation, we also build 6 plane wave indices, which depend simply on the $S|R$ translation indices. The translation index and the level of space subdivision therefore uniquely specify pointers to the arrays of the translation data. The translation indices for each box are stored along with the data on the boxes in the neighborhoods which interact with the given box. Any single translation procedure then consists in retrieval of the translation index, which are passed along with the representing vector (the of coefficients to be translated) and current level to the subroutine that performs translation (say matrix-vector multiplication) and returns the translated data. This scheme allows the theoretical minimum of the number of multiplications to be achieved but needs the algorithm to pay the price of translation data management. We found that instead of storage of all translation data it maybe computationally cheaper to manage a smaller number of translation parameters (say just the precomputed complex exponents $e^{i\alpha_{1k}}$ instead of the full matrix $e^{im\alpha_{jk}} = e^{imj\alpha_{1k}}$) and introduce one additional multiplication in the loop. If such cases were found, we used the option that results in fastest operation. In any case our translation implementation avoids costly computation of functions like complex exponents and uses a small amount of multiplications in the inner loop, which we tried to reduce as much as possible.

3.1 Algorithm for Method#0 and Method#1

Figure 3 shows main steps of the standard FMM, assuming that the preset part is performed initially. This scheme can be applied for translations using Methods #0 and #1. Here Steps 1 and 2 constitute the upward pass in the box hierarchy, Steps 3,4, and 5 form the downward pass and Steps 6 and 7 relate to final summation. The upward pass is performed for boxes in the source hierarchy, while the downward pass and final summation are performed for the evaluation hierarchy. By "near neighborhood" we mean the box itself and its immediate neighbors, which consists of 27 boxes for a box not adjacent to the boundary, and the "far neighbors", are boxes from the parent

near neighborhood (of the size of the given box), which do not belong to the close neighborhood. The number of such boxes is 189 in case the box is sufficiently separated from the boundary of the domain.

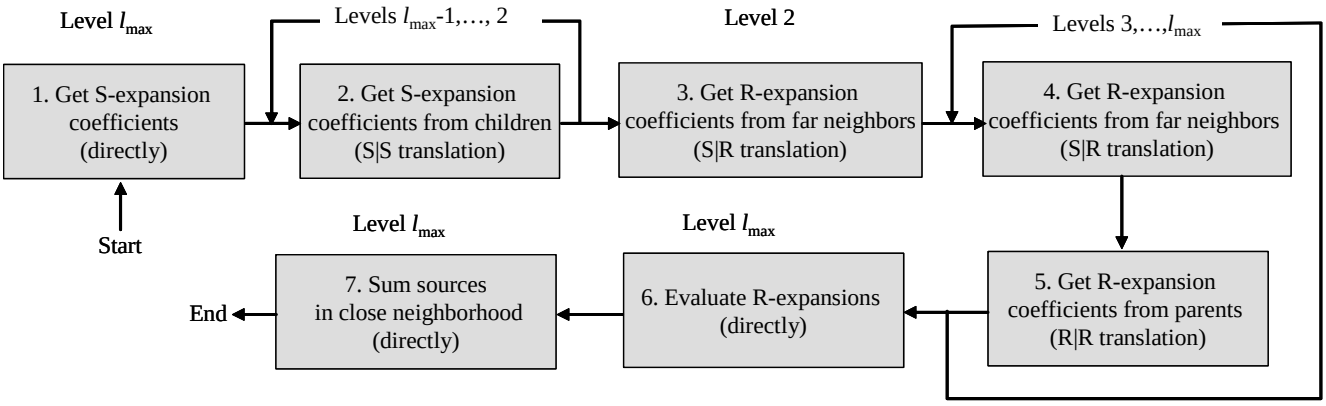

Figure 3: A flow chart of the standard FMM (Method #0 and Method #1).

To evaluate the complexity of the algorithm we consider "the worst" case, when the sources and evaluation points are distributed uniformly in a cubic domain and their number is the same, $N = M$. Each box at the finest level contains s sources, so we have

$$\frac{N}{s} = 8^{l_{\max}} \tag{63}$$

boxes at the finest level of space subdivision, $l_{\max}$.Further we will not count the costs for expansion consolidation operations and assume that the generation of the p^2 basis functions for factorization of the Green function (10) is Bp^2, and the cost of Step 1 is $Cost_{(1)} = BNp^2$. The same estimate holds for Step 6, where the local expansions need to be evaluated at N points.

The cost of the multipole-to-multipole and local-to-local translations in the above methods are the same, since both can be performed using upper triangular and transposed lower triangular matrices. If the cost of a single $S|S$-translation is $N^{(S|S)}$, then the cost of Step 2 is

$$Cost_{(2)} = N^{(S|S)} \left(8^{l_{\max}} + 8^{l_{\max}-1} + ... + 8^3\right) \approx 8^{l_{\max}} N^{(S|S)}. \tag{64}$$

where we assume that $l_{\max} \geqslant 3$ (otherwise there are no steps 4 and 5 in the algorithm). The cost of Step 3 and 4 can be combined and bounded by

$$Cost_{(3+4)} \lesssim 189 N^{(S|R)} \left(8^{l_{\max}} + 8^{l_{\max}-1} + ... + 8^3 + 8^2\right) \approx 189 \cdot 8^{l_{\max}} N^{(S|R)} \tag{65}$$

operations, where $N^{(S|R)}$ is the cost of single $S|R$ translation.

Finally the cost of direct summation, or Step 7, is $Cost_7 \approx 27 s^2 8^{l_{\max}}$, since for each box at the finest level $27s$ sources contribute to s evaluation points. Here we assume that one operation is spent for direct evaluation (we note that in practice a square root evaluation costs more than a multiplication).

Taking into account Eq. (63) and the above estimates, the total cost of the standard FMM excluding the preset step is:

$$Cost = 2BNp^2 + \frac{N}{s}\left(2N^{(S|S)} + 189 N^{(S|R)}\right) + 27Ns. \tag{66}$$

This shows that the clustering parameter s heavily influences the complexity of the method. The optimum can be easily found by differentiating the above function with respect to s. This yields an optimum value of s

$$s_{opt} = \left[\frac{1}{27}\left(2N^{(S|S)} + 189N^{(S|R)}\right)\right]^{1/2} \approx \left(7N^{(S|R)}\right)^{1/2}. \tag{67}$$

Note that this does not depend on N. We also note that for the optimum s the costs of the third and the second terms in Eq. (66) are balanced. So

$$Cost^{(opt)} = 2BNp^2 + 54Ns_{opt} \approx 2BNp^2 + 54N^2 8^{-l_{\max}^{(opt)}}, \quad l_{\max}^{(opt)} = \log_8 \frac{N}{s_{opt}}. \tag{68}$$

Neglecting $2N^{(S|S)}$ compared with $189N^{(S|R)}$ in Eq. (66), we obtain

$$Cost^{(opt)} = 2BNp^2 + 54Ns_{opt} \approx \left[2Bp^2 + 54\left(7N^{(S|R)}\right)^{1/2}\right] N. \tag{69}$$

We can substitute then $N^{(S|R)}$ given by Eqs. (42) and (43) to obtain theoretical estimates for the total cost of the FMM for Method #0 and #1:

$$Cost_0^{(opt)} \approx \left[2Bp^2 + 54\left(7N_0^{(S|R)}\right)^{1/2}\right] N \approx (2B + 143)\, Np^2, \tag{70}$$

$$Cost_1^{(opt)} \approx \left[2Bp^2 + 54\left(7N_1^{(S|R)}\right)^{1/2}\right] N \approx \left(2B + 261p^{-1/2}\right) Np^2. \tag{71}$$

This shows that both methods at large p are scaled as $O(Np^2)$, while the asymptotic constant for Method #0 is substantially smaller (if $2B \ll 143$, which is true).

Note also that the absolute error, ϵ, of the p-truncated approximation decays exponentially with p, which means that to guarantee that the total absolute error, $\epsilon_{tot} = N\epsilon$, is bounded with growing N, p must increase as $O(\log N)$. This shows that the above methods in terms of absolute error accuracy scale as $O(N \log^2 N)$ with N. However, if a weaker error norm is sufficient for solution of a particular problem (such as a relative L_2-norm error, see the next section for a more extended discussion), the the methods scale as $O(N)$.We also note that for $p^{1/2} \ll 130/B$ (which for Laplace's equation is the practically encountered value of p) Method#1 with optimum parameters scales as $O(Np^{3/2})$, while Method #0 always scales as $O(Np^2)$.

3.2 Algorithm for Method#2

Figure 4 shows the main steps of the FMM with Method#2, assuming that the preset part is performed. Here Steps 1,2,5,6, and 7 are the same as in Fig. 3, while Steps 3 and 4 involving the multipole-to-local translation are different. To provide faster $S|R$ translation for each level $l = 2, ..., l_{\max}$ a loop with respect to each direction of the exponential expansion is performed. This is characterized by the "Plane wave index".

Within this loop, first, we make a pass over all source boxes to convert multipole expansions to the exponential representations (Step 3.1). The conversion for directions other than $\pm z$ requires rotating the reference frame to point the rotated z axis towards the direction of translation, so that the centers of the boxes being translated to lie in the region $z > 2d$ in the rotated reference frame. The rotation transform adds to the conversion cost (60) for the $\pm x$ and $\pm y$ directions. Note

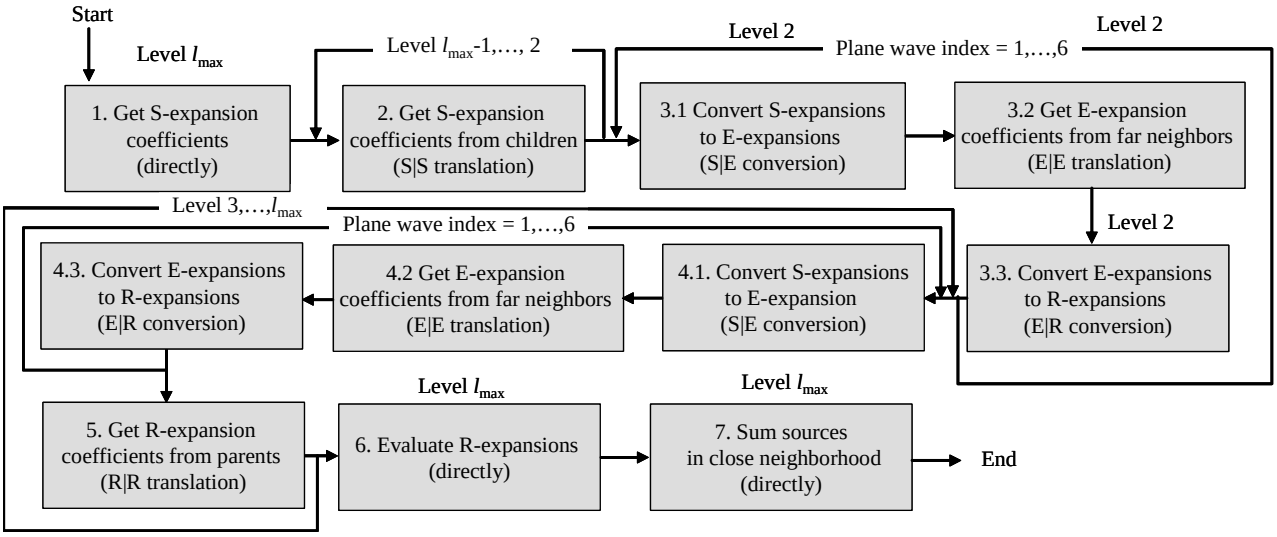

Figure 4: A flow chart of the FMM with translation performed by Method #2.

that once a transform is performed to say the x direction, the transform in the opposite direction $(-x)$ is computationally cheap, since the multipole expansion coefficients for the same function in coordinates (x, y, z) and $(\widehat{x}, \widehat{y}, \widehat{z}) = (-x, y, -z)$ are related as

$$\widehat{C}_n^m = (-1)^n C_n^{-m}, \tag{72}$$

(with the normalization $\alpha_n^m = \beta_n^m = 1$). Such a transform corresponds to the change of angles $(\theta, \varphi) \to (\pi - \theta, \pi - \varphi)$, and it is not difficult to get a generalization of this equation for arbitrary α_n^m and β_n^m using the definitions (6)-(9).

By the way, this shows that it is sufficient to perform only two $O(p^3)$ real matrix $\left\{H_n^{mm'}(\pi/2)\right\}$ rotations for axis flipping. For representations of real functions the cost per box is $2N_{Rot}(p)$ (as opposed to $4N_{Rot}(p)$ reported in the original paper [Cheng99]; however this improvement usually is small due to the much larger asymptotic constants for conversion of the expansions). Since Steps 3.1 and 4.1 must be performed for each box at corresponding levels and conversion of the multipole to the exponential expansion needs to be performed for each of the 6 translation directions $(\pm x, \pm y, \pm z)$ we evaluate the cost of these steps as

$$\begin{aligned} Cost_{(3.1+4.1)} &= (2N_{Rot} + 6N^{(S|E)}) \left(8^{l_{\max}} + 8^{l_{\max}-1} + ... + 8^3 + 8^2\right) \\ &\approx 8^{l_{\max}}(2N_{Rot} + 6N^{(S|E)}) \approx 8^{l_{\max}} \left(6\kappa + 24\sigma + \frac{8}{3}\right) p^3, \end{aligned} \tag{73}$$

where $N_{Rot}(p)$ and $N^{(S|E)}$ are evaluated according Eqs. (41) and (60). The combined cost of Step 3.3 and 4.3 is the same, due to $N^{(S|E)} = N^{(E|R)}$ (see Eq. (62)) and 2 rotations are needed to flip the axes back. The combined cost of Step 3.2 and 3.3 is given by Eq. (65), where instead of $N^{(S|R)}$ we need to use $N^{(E|E)}$ from Eq. (56), which is valid for the real case. We obtain the total cost of the FMM for Method #2, where the $S|S$ and the $R|R$ translations are performed by Method #1 (which is faster than Method #0 for $p \geqslant 4$),

$$Cost = 2BNp^2 + \frac{N}{s}\left[2N^{(S|S)} + 2(2N_{Rot} + 6N^{(S|E)}) + 189N^{(E|E)}\right] + 27Ns. \tag{74}$$

The same evaluation step cost is taken as for the other methods. Here we do not neglect the $O(p^2)$ cost of the $E|E$ translations compared to the other terms, since the asymptotic constant 189 is large.

This yields the optimum value of s

$$\begin{aligned} s_{opt} &= \left\{ \frac{1}{27} \left[2N^{(S|S)} + 2(2N_{Rot} + 6N^{(S|E)}) + 189N^{(E|E)} \right] \right\}^{1/2} \\ &\approx \left\{ \frac{2p^2}{27} \left[\left(6\kappa + 24\sigma + \frac{17}{3} \right) p + 189\sigma \right] \right\}^{1/2}. \end{aligned} \tag{75}$$

So the cost of the optimized FMM which employs Method #2 is

$$\begin{aligned} Cost_2^{(opt)} &= 2BNp^2 + 54Ns_{opt} \approx \left\{ 2Bp^2 + 54 \left\{ \frac{2p^2}{27} \left[\left(6\kappa + 24\sigma + \frac{17}{3} \right) p + 189\sigma \right] \right\}^{1/2} \right\} N \\ &\approx \left[2B + 15p^{-1/2} \left(6\kappa + 24\sigma + 6 + 189\sigma p^{-1} \right)^{1/2} \right] Np^2. \end{aligned} \tag{76}$$

In the last result we rounded constants to the closest integers.

3.3 Comparison of theoretical complexities

An explicit evaluation of the costs is not given in the papers [Greengard97, Cheng99], and it is claimed that method #2 is always faster than method #1. As the following analysis shows, this need not be the case. A first look at Eqs. (71) and (76) shows that both methods #1 and #2, have the same asymptotic behavior for fixed N and $p \to \infty$, which is $\sim 2BNp^2$. This is due to for computations with extremely large p, not the translations, but generation and evaluation of expansions is the most costly step, which both methods treat by the same way. However this case is not practical, since a good accuracy can be achieved at relatively small p, where, in fact Steps 1 and 6 in the algorithms are less costly than translations (while they may contribute several percents into the total cost).

Let us define an intermediate asymptotic region of p by such way that the cost of other operations much larger than the cost of expansions/evaluations. For methods #1 and #2, respectively, this can respectively be written as

$$1): \quad p^{1/2} \ll \frac{130}{B}, \quad 2): \quad p^{1/2} \ll \frac{15}{2B} \left(6\kappa + 24\sigma + 6 + 189\sigma p^{-1} \right)^{1/2}. \tag{77}$$

Now, if p satisfies Eq. (77), and computations are performed with optimal settings we can determine that

$$\eta = \frac{Cost_2^{(opt)}}{Cost_1^{(opt)}} \approx \frac{15}{261} \left(6\kappa + 24\sigma + 6 + 189\sigma p^{-1} \right)^{1/2} \approx 0.06 \left(6\kappa + 24\sigma + 6 + 189\sigma p^{-1} \right)^{1/2}. \tag{78}$$

Our numerical tests show that constants κ and σ are substantially larger than unity to have both methods performing with the same accuracy. This means that the relative costs of the two methods is likely to be similar. However, we can provide a bound for the best improvement that can be achieved by use of Method 2, by making assumptions that are favorable to it. This occurs if we set $\kappa = \sigma = 1$ and neglect $189\sigma p^{-1}$ (which can also be significant). In this case the right hand side of Eq. (78) will be $0.06 \cdot 36^{1/2} \approx 0.36$. So, perhaps, the largest improvement one can expect to achieve with Method #2 is a three-fold increase in the speed for high accuracy.

3.4 Determination of asymptotic constants based on absolute errors

Before validating these estimates in actual FMM performance tests, we evaluate the constants κ and σ by evaluating the error achieved in the representations. Since the accuracy of each method depends on p and the order of quadrature (length of the exponential expansion), we can compare two different methods only in terms of the error they produce. The error should be the same, or at least of the same order. We performed some small numerical translation tests, the results of which are plotted in Figs. 5 and 6.

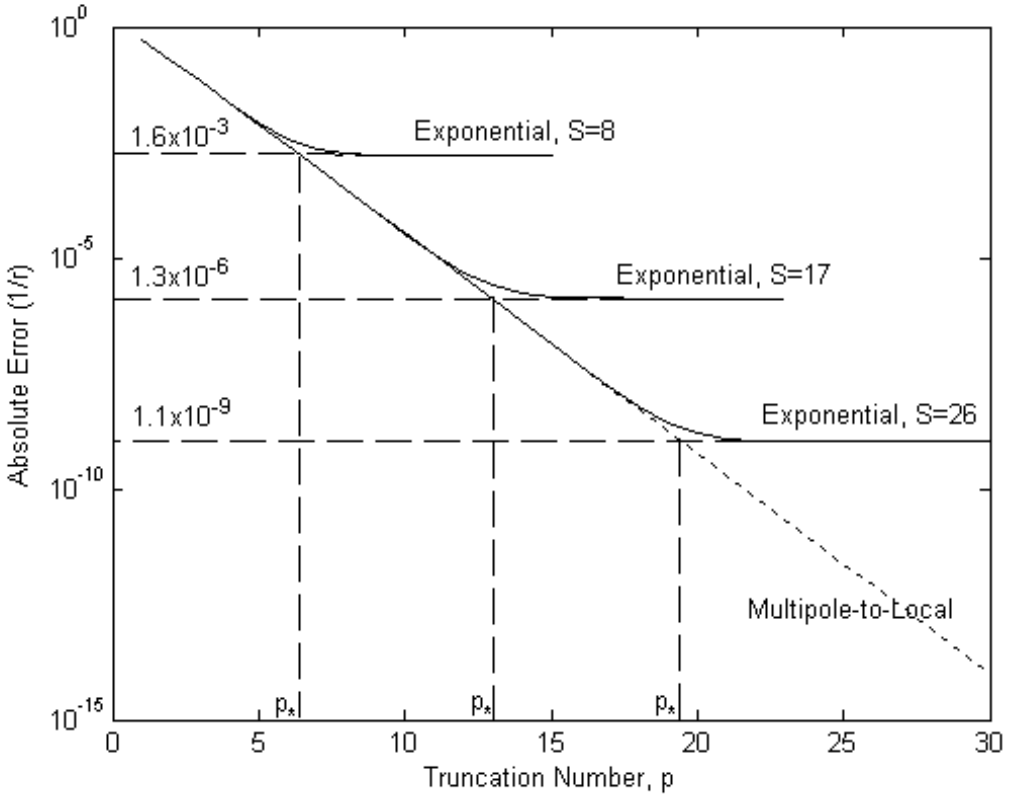

Figure 5: Absolute maximum error of expansion/translation for a single source ($1/|\mathbf{r}-\mathbf{r}_0|$) located at $\mathbf{r}_0 = (0, 0, 0.5)$ and evaluated for 101 points $\mathbf{r} = (0, 0, z)$, $1.5 \leqslant z \leqslant 2.5$. The solid curves show the error behavior for Method #2 with the order of quadrature $S(\epsilon)$ shown near the curves as function of the truncation number p at fixed $S(\epsilon)$. The dotted curve shows the error for Method #1. The dashed lines show the error asymptotics and maximum truncation numbers for which the error cannot be improved for given $S(\epsilon)$.

Figure 5 illustrates the absolute error of expansion/translation of the function $1/r$ and corresponds to the worst case. Here the source is located in the closest position to the box where the translation should be performed. The absolute error, ϵ, was evaluated by comparison of the exact solution with approximate solution, which was obtained by building of p-truncated multipole expansion for a box centered at the origin of the reference frame followed by multipole-to-local translation to the center of the evaluation box and evaluation for several point located on the axis

of translation. For Method #2 the multipole-to-local translation was performed by conversion of the multipole to exponential expansion, exponential translation, conversion of the exponential to local representation and evaluation. We found that in our tests the minimum ϵ that can be achieved using Method #2 coincides with the reported values $\epsilon = 1.6 \cdot 10^{-3}$, $1.3 \cdot 10^{-6}$ and $1.1 \cdot 10^{-9}$ for $S(\epsilon) = 8, 17$, and 26, respectively [Greengard97]. These errors cannot be improved by increasing the truncation number for fixed quadrature. On the other hand, the error exponentially decays at increasing p when using Method #1 (or Method #0, which in terms of error is the same as Method #1). So there is no need to perform computations with $p > p_*(\epsilon)$ using Method #1 as this method should be compared with Method #2 in terms of achieving the same accuracy. The "critical" values $p_*(\epsilon)$ are determined by the quality of the Method #2 quadrature and are 6,13, and 19 for the case considered. This allows us evaluate constants κ and σ based on their definitions Eq. (53) by substituting there found $p_*(\epsilon)$ and given quadrature parameters $S(\epsilon)$ and $S_{\exp}$. The results together with parameter η from Eq. (78) are presented in the following table.

$p_*(\epsilon)$	$S(\epsilon)$	$S_{\exp}$	κ	σ	η
6	8	104	1.33	2.89	0.79
13	17	516	1.31	3.05	0.69
19	26	1340	1.37	3.71	0.71

Table 1: Complexity constants.

These results show that theoretically Method #2 should perform faster than Method #1 for the given maximum absolute accuracy. It is also noticeable that in all cases parameter $\eta \approx 0.7$-0.8, which means that theoretically expected increase in speed when using Method#2 is 20-30% for the given p and quadrature. This substantially differs from the expectations, which one can have based on the analysis presented in [Greengard97, Cheng99], where the cost of the multipole-to-local translations is incorrectly estimated as $189 \cdot 3p^3$, while the cost of of the same operations is described as $20p^3 + 189p^2$. The ratio of this cost to the correct one presented here is about $0.035 + 1/(3p)$, which is one order in magnitude less than the estimate derived here. This might have lead the authors to an incorrect claim that method #2 is always faster.

The actual situation may be even worse for Method #2. Indeed, the maximum theoretical error is achieved only for the case of the closest box that is separated by one box. Further the evaluation box, lesser $p_*(\epsilon)$, at which the accuracy of the two methods is the same. Note that while the absolute error is smaller for the remote boxes from the set of 189 boxes, their number is substantial and their total contribution to the error can be comparable or greater than the maximum error for a single box. Reduction of $p_*(\epsilon)$ is is illustrated by Fig. 6, where the computed error for the evaluation box separated from the source box by two boxes is presented. Here we assume that the source is located at the center of the source box and found the maximum error as before. These results show that $p_*(\epsilon)$ is 4,8, and 12, for $S(\epsilon) = 8, 17$, and 26, respectively. Computation of the complexity constants based on this evaluation is provided in the following table.

$p_*(\epsilon)$	$S(\epsilon)$	$S_{\exp}$	κ	σ	η
4	8	104	2	6.5	1.32
8	17	516	2.13	8.1	1.20
12	26	1340	2.17	9.3	1.18

Table 2: Complexity constants.

This shows that for these numbers Method #1 in fact should perform better (theoretically (!))

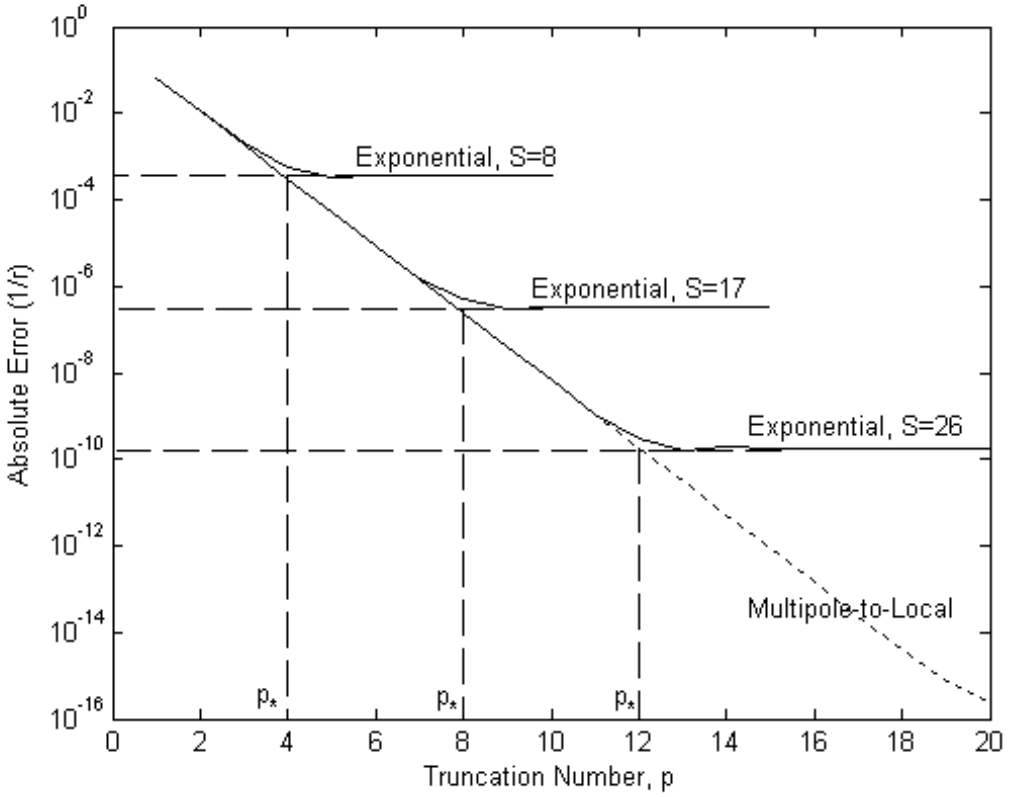

Figure 6: Absolute maximum error of expansion/translation for a single source ($1/|\mathbf{r}-\mathbf{r}_0|$) located at $\mathbf{r}_0 = (0,0,0)$ and evaluated for 101 points $\mathbf{r} = (0,0,z)$, $2.5 \leqslant z \leqslant 3.5$. Notations are the same as in Fig. 5.

than Method #2. The differences are not very large: the same 20%-30%, but now in favor of Method #1. For more accurate evaluations, which adds also some practical issues, we performed full FMM tests, where the actual errors were measured for random point distribution (see the section below).

3.5 Memory Complexity

In all methods considered there are three major memory consuming factors. First, the "book-keeping" related to the storage of data structure that include lists of neighbors, etc. Second, the memory is required to store precomputed translation data (e.g. rotation matrices), and, third, we need to store representing vectors during the upward and downward passes. The storage due to these factors is usually very different, and, normally the major memory is required for third item, namely the length of the representing vectors. In terms of memory the worst situation occurs when there are no empty boxes, and we need to store the function representations for each box (uniform data distribution inside a cube). Note that 7/8 of all boxes are at the finest level of space subdivision, so for simplicity we can neglect the storage for the other boxes, especially since we can deallocate from memory some data after they are used.

To store a list of 27 neighbors for $2^{l_{\max}}$ boxes, assuming that the index of each box is a 4 byte

integer we need

$$M^{(1)} = 27 \cdot 2^{l_{\max}} \cdot 4 \tag{79}$$

bytes of storage.

The memory required for storage of translation data depends on the method used. Since the most consuming part is related to multipole-to-local translations we will neglect local-to-local and multipole-to-multipole translation data in the memory storage. Assume that for Method #0 we store all S|R-translation matrices, which number is 343 for each level, and which contain p^4 real entries occupying 8 bytes of memory each. In this case we need storage

$$M_0^{(2)} = 343p^4 \cdot (l_{\max} - 1) \cdot 8. \tag{80}$$

For Method #1 there are 15 different translation distances for each level and the coaxial S|R translation matrix requires $\approx p^3/3$ storage of real numbers. In addition there are 48 different non-zero rotation angles β. Taking into account that storage of the rotation matrix requires $\approx 2p^3/3$ real numbers, and the rotation matrices are the same for all levels, we have the memory complexity

$$M_1^{(2)} = 15\frac{p^3}{3} \cdot (l_{\max} - 1) \cdot 8 + 48\frac{2p^3}{3} \cdot 8 = 8p^3 \left(5l_{\max} + 27\right). \tag{81}$$

For Method #2 the storage of precomputed translation data can be done with memory complexity

$$M_2^{(2)} = 8S\left(\epsilon\right)\left(3S_{\exp} + \frac{p^2}{2}\right)\left(l_{\max} - 1\right) + \frac{2p^3}{3} \cdot 8. \tag{82}$$

Finally, the storage of representations of length p^2 for Methods #0 and #1 at level $l_{\max}$ requires

$$M_{0,1}^{(3)} = 2 \cdot 2^{l_{\max}} p^2 \cdot 8 \tag{83}$$

bytes, where factor 2 appears due to storage of both multipole and local coefficients is needed. For Method #2 we need at least

$$M_2^{(3)} = 2^{l_{\max}} S_{\exp} \cdot 8 + 2 \cdot 2^{l_{\max}} p^2 \cdot 8 \tag{84}$$

bytes, where the first number is related to functions representation via exponential expansion, and the second number is the minimum needed to store the multipole and local expansions.

For Methods #1 and #2 the storage of translation data is much smaller than the storage of representations, while for Method #0 this can be substantial. We also can neglect the storage of the neighbor list. With such simplifications and using Eq. (53) we can evaluate the memory requirements for Methods #1 and #2 as follows:

$$\begin{aligned} M_1 &\approx M_1^{(3)} = 16 \cdot 2^{l_{\max}} p^2, \\ M_2 &\approx M_2^{(3)} = 16 \cdot 2^{l_{\max}} p^2 \left(1 + \frac{1}{2}\sigma\right). \end{aligned} \tag{85}$$

This shows that e.g. for $\sigma = 3$ the ratio $M_2/M_1 = 2.5$. This shows a substantial increase of the memory complexity for Method #2 compared to Method #1. Thus a strong advantage for Method #1 relative to Method #2 is that for a given memory size it can fit a larger problem.

3.6 Possible Reductions of Asymptotic Constants

We note that there are several opportunities to reduce the number of operations in conversion for Method #2. One of them is related to the observation that the conversions S|E and E|R involve sums of complex exponents, and, in fact are Fourier transforms of periodic functions. One can exploit some ideas of the FFT (the Danielson-Lanczos lemma) to reduce the complexity of the direct summation. We tried one-step reductions using this lemma and found that the overall speed can be improved by a few percents. Another opportunity is to use some symmetries to reduce the asymptotic constant 189. Greengard and Rokhlin [Greengard97] mention that there may be such possibilities, but do not provide them, and their results in [Greengard97] and [Cheng99] appear to not use such symmetries. Some savings can be done for cases of low p when in some places of the code single precision real numbers can be used instead of double precision without substantial loss of accuracy. We also should mention, that if some symmetries are used for translations in Method #2, then symmetries for Methods #0 and #1 also should be considered more carefully. E.g. the translation matrix can be decomposed to a sum of two matrices (half of elements of each are zeros) so the translation with vector $\mathbf{t}$ can be done using a sum of these matrices, while translation in direction $-\mathbf{t}$ can be done using the difference. In the numerical tests provided below we did not use the above-mentioned opportunities.

4 Numerical tests

The problem that was solved in the numerical tests is related to potential N-particle interaction (e.g. gravity or electrostatic forces) and is the same as in [Cheng99]. Here we compute

$$\phi(\mathbf{r}_j) = \sum_{i=1, i\neq j}^{N} q_i G(\mathbf{r}_i, \mathbf{r}_j), \quad j = 1, ..., N, \tag{86}$$

where q_i are the source intensities, which in tests were set all equal. The sources were randomly uniformly distributed inside a cube (say, the unit cube, since the Laplace equation does not have its own scale and all results can be scaled to the domain of arbitrary size).

The code was realized in Fortran 90 and compiled using the Compaq 6.5 Fortran compiler. All computations were performed in double precision. In the tests we used the version of the FMM for real functions. The CPU time measurements were conducted on a 3.2 GHz dual Intel Xeon processor with 3.5 GB RAM. Some particulars of our implementation are mentioned in the previous section. Different translation methods were implemented in the same programming environment and most subroutines used (except of specific translation subroutines) were the same, which enables us to conduct a comparison of performance of those methods. We also paid attention to efficiency of implementation trying to use the fastest computational algorithms for each method. We also compared our implementation with the "cross-over" points of other implementations reported in the literature.

4.1 Computation of errors

To compare the FMM utilizing different translation methods first we introduce the measures for the errors. If there are M evaluation points in the domain, the absolute errors in the L_∞ and L_2

norms are

$$\epsilon_\infty^{(abs)} = \max_{j=1,\ldots,M} \left|\phi_{exact}(\mathbf{r}_j) - \phi_{approx}(\mathbf{r}_j)\right|, \quad \epsilon_2^{(abs)} \tag{87}$$
$$= \left[\frac{1}{M}\sum_{j=1}^{M} \left|\phi_{exact}(\mathbf{r}_j) - \phi_{approx}(\mathbf{r}_j)\right|^2\right]^{1/2},$$

where $\phi_{exact}(\mathbf{r})$ and $\phi_{approx}(\mathbf{r})$ are the exact and approximate solutions of the problem. The absolute errors, however, are not representative, since they depend on the scaling of the source intensities. To introduce scale independent relative errors, we can select some representative scale, e.g. some norm of the exact solution. The scale can be selected as the L_2 norm of function $\phi_{exact}(\mathbf{r})$:

$$\|\phi_{exact}(\mathbf{r})\|_2 = \left[\frac{1}{M}\sum_{j=1}^{M} |\phi_{exact}(\mathbf{r}_j)|^2\right]^{1/2}. \tag{88}$$

So the absolute errors were related to this scale:

$$\epsilon_\infty = \frac{\epsilon_\infty^{(abs)}}{\|\phi_{exact}(\mathbf{r})\|_2}, \quad \epsilon_2 = \frac{\epsilon_2^{(abs)}}{\|\phi_{exact}(\mathbf{r})\|_2}. \tag{89}$$

The exact solution was computed by straightforward summation of the source potentials (86). This method is acceptable for relatively low M, while for larger M the computations become unacceptably slow, and the error can be measured by evaluation of the errors at smaller number of the evaluation points. Figure 7 shows comparison of the ϵ_∞ and ϵ_2 errors computed over the full set of the evaluation points and over the first 100 points. It is seen that the errors in the L_2 norm are almost the same for these two cases, while a difference is observed for the errors in the L_∞ norm. While this difference seems is not very large and further we will use smaller number of the evaluation points to estimate the error (especially when we concern about the orders of magnitude), we should state that ϵ_∞ computed over first 100 random points represents some probabilistic measure and not exactly ϵ_∞, which can be larger.

4.2 Error dependence

The errors specified above depend on several factors, including the number and location of the sources, the translation method used, the truncation number, and the number of levels in the FMM. Since the accuracy of Methods #0 and #1 is the same in terms of error dependences, only Methods #1 and #2 are compared.

For the fixed distribution of N sources we can consider dependence of the errors on the truncation number p and the maximum level of space subdivision $l_{\max}$ used in the FMM. Concerning the latter dependence we note that this dependence is much weaker than the error dependence on the truncation number. So for any level and given truncation number we have approximately the same error. On the other hand the dependence of the error on p is strong and qualitatively different for Method #1 and Method #2. Figure 8 shows dependences of ϵ_2 and ϵ_∞ on p for fixed N and $l_{\max}$. Qualitatively he behavior is similar to that for absolute errors shown in Figs. 5 and 6. We note that p_* depends on the norm in which the error is measured. This "critical" truncation number is larger if ϵ_∞ is used. Based on the L_2 norm, which is confirmed by the performance study the critical truncation numbers are $p_* = 4, 9$, and 19. For these numbers the following theoretical complexity constants are expected.

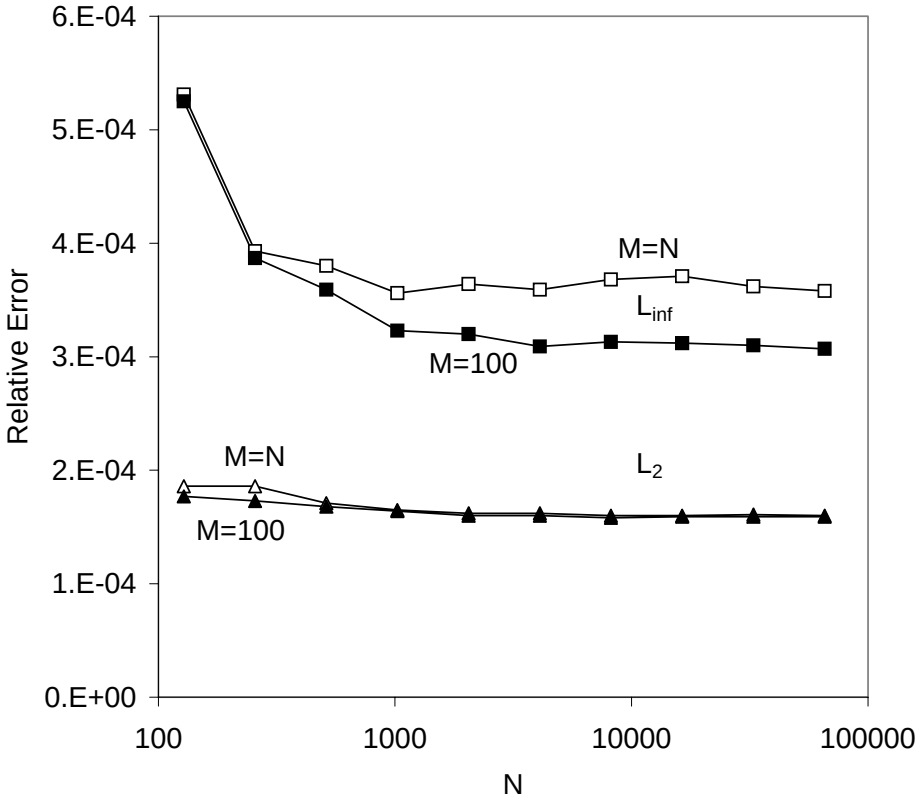

Figure 7: The relative errors ϵ_∞ (marked as L_{inf}) and ϵ_2 (marked as L_2) computed according Eq. (89) for the full set of evaluation points ($M = N$) and for the first 100 evaluation points ($M = 100$). Computations performed for translation scheme using exponential expansions, $p = 6$, $l_{\max} = 3$, sources of equal intensity are randomly distributed inside a cube.

Figure 9 illustrates dependence of the error on the size of the problem N. According the theory, the absolute error of the multipole expansions decays exponentially at the growing p. Assuming that each source contributes equally to the total absolute error we can state then that to provide a constant absolute error we should have $p \sim \log N$. For computation of the solid curve shown in Fig. 9 we used the Method #1 with $p = \log_2 N + 2$. It is seen that the absolute error $\epsilon_2^{(abs)}$ does not increase in this case, that is consistent with the theory. Figure 9 also shows that for the increasing N the norm of function $\|\phi_{exact}(\mathbf{r})\|_2$ grow proportionally to N. The same is true for the absolute error $\epsilon_2^{(abs)}$ if computations are performed at constant p. This means that the relative error ϵ_2 stays stably within some bound at constant p. So if the purpose of computations is to obtain solution with some prescribed relative error, then this can be done using constant p (in which case the FMM is an $O(N)$ algorithm). Figure 9 illustrates this fact for computations using Method #1 and Method #2, where for Method #2 we selected $p = p_*$ (for the relative L_2 error as provided by Fig. 8). Note that we also computed errors $\epsilon_\infty^{(abs)}$ and ϵ_∞ for the same settings. While these errors are several times higher than $\epsilon_2^{(abs)}$ and ϵ_2, respectively, their qualitative behavior is the same as for the errors in the L_2 norm. While some discrepancy is observed for errors computed by the two translation methods at $p = 9$, errors $\epsilon_\infty^{(abs)}$ and ϵ_∞ are almost the same for these methods. This is

$p_*(\epsilon)$	$S(\epsilon)$	S_{exp}	κ	σ	η
4	8	104	2	6.5	1.32
9	17	516	1.89	6.4	1.05
19	26	1340	1.37	3.7	0.71

Table 3: Complexity constants.

explainable by Fig. 8.

4.3 Optimization

We formulate the optimization problem as follows. The optimum performance means the setting which achieves the minimum of the CPU time required for computation of a given problem (a fixed source distribution) at the constraint that the error is smaller than some prescribed error. We measured the CPU time only for the "run" time of the algorithm, which, first, has the major contribution to the overall CPU time, compared to the preset step (which is also sensitive to the way of implementation of data structures). By this definition the optimum performance can depend on the norm in which the error is measured. Once the source distribution is fixed only two parameters, p and $l_{\max}$ can be varied. When using Method #2 with some fixed order quadrature, we understand that the error cannot be arbitrary, but it should be not less than the minimum error provided by this order of quadrature. In this case to compare performance of the FMM with different translation methods we set $p = p_*$ and then optimize $l_{\max}$ to achieve the minimum CPU time for Method #2. This determines the error of the method at this optimum $l_{\max}$. Then for Method #0 and Method #1 we solve an optimization problem to determine p (which, presumably, is the same as p_*) and $l_{\max}$ that provide the same error as the error of Method #2 (or close to that – the criteria for computations with the same error in practice can be relaxed assuming that, say, two errors differ not more than 30% or 50% they are the same).

Figure 10 illustrates selection of the optimum maximum level. Here first CPU time was measured for Method #2 at different $l_{\max}$, which shows that for given $p = 9$ ($p_* = 9$ for the relative error in the L_2 norm) the optimum $l_{\max} = 4$. At this optimum we found that $\epsilon_2 = 1.22 \cdot 10^{-7}$ (measured over first 100 evaluation points). Further we did similar tests using Method #0 and Method #2 for $p = 9$ and found that for Method #0 the optimum $l_{\max} = 3$, at which $\epsilon_2 = 7.15 \cdot 10^{-8}$, while for Method #1 the optimum achieves at $l_{\max} = 4$, in which case $\epsilon_2 = 7.44 \cdot 10^{-8}$. Then we performed computations with $p = 8$ and found that the errors in this case ($\epsilon_2 = 3.01 \cdot 10^{-7}$ for Method #1 and $\epsilon_2 = 2.77 \cdot 10^{-7}$ for Method #0 at optimum levels) are substantially larger (more than 50%) than for Method #2 and $p = 9$. So we conclude that the optimal settings for computations with prescribed error $\epsilon_2 \sim 10^{-7}$ for all methods are $p = 9$ and $l_{\max} = 3$ for Methods #0 and $l_{\max} = 4$ for Method #1 and #2 .

To conduct further comparative performance study, we conducted similar optimization for all reported cases. Some insight to optimization problem at constant p is provided in the theoretical section, where it is shown that in this case the optimum theoretically should be achieved at the same values of the clustering parameter s. We checked this with the current code and found that this is a valid statement. In any case we varied the clustering parameter, which affects only $l_{\max}$ to be sure that indeed the optimum setting is provided for every random distribution.

It is also noticeable that figures in our implementation are close to the theoretical predictions. For example, for $p = 9$, Eqs. (67), (42), and (75) predict $s_{opt} = 130$ for Method #1 and $s_{opt} = 128$ for Method #2 (which is almost the same, since this determines the same $l_{\max}$). we found that computations with $s = 128$ (we varied this parameter by doubling) are indeed optimal for the

range of levels $l_{\max} = 2, ..., 6$. Fig. 10 also shows that Methods #1 and #2 performs almost the same. This agrees with the theoretical estimate (in the previous section we found that the ratio of methods costs is $\eta = 1.05$, which is very close to 1 for case $p = 9$, $S(\epsilon) = 17$.

4.4 Performance study

We conducted performance study for three cases of "low", "moderate", and "large" truncation numbers corresponding to the accuracy provided by the quadrature formula using different translation methods. These correspond to the cases considered in [Greengard97]. As we mentioned above the CPU time was measured only for the "run" part of the algorithm, which opposed to the preset step should be performed only once if the point distribution does not change.

4.4.1 Low accuracy

In this case when using Method #2 we employed the quadrature formula of order 8. Figure 11 illustrates error dependences on N for the optimized FMM with different translation methods and different truncation numbers. This shows that when using Method #2 we have $p_* = 4$.

Indeed, in this case computations with $p = 4$ and $p = 5$ yield the same order of the error, and there is no improvement in accuracy for larger p both for the ϵ_2 and ϵ_∞ errors. On the other hand, the errors obtained when using Method #1 while slightly higher are about the same as the errors for Method #2 for $p = 4$. Computations with $p = 5$ and Method #1 provide substantially smaller errors compared to computations using $p = 4$ and $p = 5$ and Method #2.

Figure 12 illustrates the performance of the optimized FMM using different translation methods. As it is seen at low truncation numbers all the methods require about the same time to compute the same problem. Method #1 at $p = 4$ is a bit faster than the other methods. Increase of p to $p = 5$ increases the CPU time and, in fact makes Method #1 a bit slower than Method #2. This is consistent with the above theory, which predicts the ratio of speeds $\eta > 1$ for $p = 4$ and $\eta < 1$ for $p = 5$. This figure also shows that our implementation of the FMM is pretty efficient, since all the methods perform faster than the direct, or straightforward, summation for $N \geqslant 10^3$ (in fact, the break-even point, N_*, is 2-3 times smaller than $N = 10^3$; for $p = 4$ and Method #0 we found $N_* = 380$, Method #1 yields $N_* = 320$ for the same p, and Method #2 for $p = 4$ and the $S(\epsilon) = 8$ has $N_* = 430$, which is somehow lower than $N_* = 750$ reported in [Cheng99] for this method with the same accuracy (presumably larger p was used in the cited reference, also the break-even point depends on the implementation; we also note the folowing differences: in the cited paper computations were made with a single precision, and it is not clear whether the total CPU time or the CPU time for the "run" part of the FMM is measured).

4.4.2 Moderate accuracy

In this case when using Method #2 we employed the quadrature formula of order 17. Figure 13 illustrates error dependences on N for the optimized FMM similarly to Fig. 11. This shows that when using Method #2 we have $p_* = 9$. Indeed, the use of $p = 10$ does not improve the ϵ_2 error for Method #2. At the same time computations with $p = 9$ using Method #1 have errors ϵ_2 smaller than those for Method #2, while for $p = 10$ the errors for Method #1 are several times smaller than that for Method #2.

Figure 14 shows the CPU time measurements for the optimized FMM employing different translation schemes. This comparison shows that at higher N Method #1 and Method #2 require about the same time for computation of the same problem with the same accuracy, while Method

#1 is slightly faster at lower N and slightly slower at larger N. The difference of speed of Method #1 and Method #2 is really small. For example, computation of $N = 2^{20} \approx 10^6$ problem takes 112 seconds for Method #1, while 105 seconds for Method #2, which is just 6% better. Such differences normally should not be taken into account, since different implementation may change the situation to the opposite (we tried to implement each method in the fastest way). It is interesting to note that for $p = 9$ in our implementation the break-even points are about $N_* = 1800, 900$, and 1100 for Method #0, #1, and #2, respectively. Note that the break-even point $N_* = 1500$ reported in [Cheng99] for Method #2.

Being $O(p^4)$ Method #0 at high N slows down several times compared to the $O(p^3)$ translation methods, which is an expected result. So, while this method is comparable with the other two methods for low accuracy computations, for higher accuracy it loses the competition at least to Method #1, which has just the same accuracy, but is substantially faster. We note that these results show that the improvement achieved by method 1 are better than that reported in the original paper [White96], and close to the predicted improvement. Our results for low accuracy from the previous subsection also show that there is no any advantage for using Method #0 even in the case $p = 4$ (however we checked that for $p = 3$ Method #0 is in some cases slightly faster than the $O(p^3)$ methods).

4.4.3 High accuracy

In this case when using Method #2 we employed the quadrature formula of order 26. Figure 15 illustrates error dependences on N for the optimized FMM similarly to Fig. 11. This shows that when using Method #2 we have $p_* = 19$. Indeed, the use of $p = 20$ does not improve ϵ_2 errors for Method #2. Error ϵ_2 for Method #1 with $p = 19$ is comparable with ϵ_2 for Method #2 at lower N and much smaller at larger N (so smaller p_* can be used in this case). Fig. 19 shows that the use of $p_* = 16$ provides larger error, while $p_* = 17$ can be used for large N.

Figure 16 is similar to Figs.12 and 14. As in the previous figures we can see that the FMM using Method #1 and Method #2 spend approximately the same time for solving the same problem with the same accuracy. At lower N Method #1 is slightly faster than Method #2, while for larger N Method #2 is faster than Method #1. For example to compute the problem with $N = 2^{13} \approx 10^4$ the FMM run times for Methods #1 ($p = 19$) and #2 are 1.2 and 1.25 seconds, respectively. For $N = 2^{21} \approx 2 \cdot 10^6$ the CPU times for Methods #1 ($p = 19$) and #2 are 665 and 502 seconds, respectively, i.e. Method #2 performs 25% faster than Method #1 (for larger N the ratio varies from 10 to 35%, since the optimal level for Method #2 should be lower than for Method #1, but since the level can be only an integer, some non-optimal performance cannot be avoided, which is consistent with the 29% difference predicted by the theory). The use of variable p improves the speed of Method #1 at larger N, in which case Method #2's performance advantage is at most 25% over Method #1 for any N). We also note that the break-even point with the direct method increases in our implementation up to approximately $N_* = 2500$ for both Methods #1 and #2. This coincides with the break-even point $N_* = 2500$ reported in [Cheng99]. Concerning Method #0 it should be noted, that it is not competitive for computations with high accuracy (also notice its high break-even point for Method #0, $N_* > 10000$).

Figure 17 demonstrates the CPU time required for the performance of the FMM (the same data as in Fig. 16) and for presetting of the FMM run (computation related to data structure and precomputation of the translation operators). The latter step should be performed only once as the data points do not change, while the former step can be performed many times (e.g. in iterative solvers). It is seen that the preset step normally takes only several percents of the FMM run time and the cost of this step is approximately the same for Methods #1 and #2. The preset time

mainly depends on the maximum level of space subdivision in the FMM, that is why the graph looks like a staircase function, where the jumps occur when $l_{\max}$ changes (in the case reported for optimal performance $l_{\max}$ changed from 2 to 5 and those changes are clearly seen on the graph for the FMM preset time). Finally we note that the CPU time to preset the FMM in the case of high accuracy computations at large N is about two orders of magnitude of the run time.

4.5 Discussion

The results of the performance study show that while the FMM using translation Method #1 or translation Method #2 can be slightly faster, the both $O(p^3)$ methods provide the same order speed and accuracy. The behavior observed in practice is consistent with the theoretical estimates, which can be used to predict relative algorithm performance.

Another issue affecting the performance is that the memory required for Method #2 is larger than for Method #1. The increase of the memory for Method #2 compared to Method #1 is about $1 + \sigma/2$ times (with σ given in the tables in the previous sections). This affects not only the issue of RAM availability for the task and limits the size of the problems that can be solved on a given CPU, but also affects the speed of retrieval and storage of data required in the process of algorithm execution (we checked this by profiling of the programs). Indeed the multipole-to-local translation operation can be performed millions times and in this count increase of the local data size can be significant compared to the overall count of operations, which may modify the asymptotic constants for each method.

5 Conclusions

We implemented the FMM with the original $O(p^4)$ and two $O(p^3)$ translation schemes (the "point-and-shoot" method of [White96] and the exponential translation scheme of [Greengard97, Cheng99]) in the same programming environment and conducted comparative performance study in terms of the speed of the FMM that achieves the same prescribed error. This study showed that:

1). Low relative errors in the L_2 norm (evaluated over 100 random points) can be achieved at relatively low truncation numbers p.

2). For low accuracy computations ($\epsilon_2 \sim 10^{-4}$) all the translation schemes have almost the same performance, while the "point-and-shoot" method is slightly faster then the other methods considered here. For this accuracy, we note that treecodes are competitive as well, and may be the method of choice.

3) For moderate accuracy ($\epsilon_2 \sim 10^{-7}$) the original FMM ($O(p^4)$) is several times slower than $O(p^3)$ methods tried, which have about the same performance. For high accuracy ($\epsilon_2 \sim 10^{-10}$) the $O(p^4)$ method is substantially slower than the $O(p^3)$ translation methods. In this case both $O(p^3)$ methods perform approximately with the same speed. For high accuracy computations the "point-and-shoot" method is slightly faster for lower N ($N \lesssim 10^4$), while the method using the exponential expansions for the multipole-to-local translations is faster for higher N ($10^5 \leqslant N \leqslant 10^6$). The difference between the speed of the methods performing with the same accuracy (for maximum accuracy achievable by a given quadrature) is within 25% with respect to the speed of the "point-and-shoot" method for computed benchmark cases.

4) The memory required for the exponential translation is significantly higher.

5) The observed performance is explained by several factors. First, the number of terms in the multipole expansions p^2 that can be used to achieve the same errors as provided by the quadrature order for the exponential expansion are several times smaller than the number of terms in the exponential expansion, $S_{\exp}$. Second, more accurate complexity estimates show that previously

reported complexity constants for the method using the exponential expansion are underestimated. Third, there exist a necessary overhead which slow down the method based on the exponential expansion due to $S_{\exp} > p^2$. Finally, the memory required for Method #2 is larger than for Method #1.

6 Acknowledgments

Partial support of NSF awards 0086075 and 0219681 is gratefully acknowledged. We thank Dr. Peter Teuben for discussions regarding tree codes.

References

[Abramowitz65] M. Abramowitz, and I. A. Stegun, Handbook of Mathematical Functions, Dover Publications, 1965.

[Barnes86] J. Barnes and P. Hut, A hierarchical $O(N \log N)$ force-calculation algorithm, Nature, 324, 1986, 446-449.

[Carrier88] J. Carrier, L. Greengard, and V. Rokhlin, A fast adaptive multipole algorithm for particle simulations, SIAM J. Sci. Statist. Comput., 9, 1988, 669-686.

[Cheng99] H. Cheng, L. Greengard, and V. Rokhlin, A fast adaptive multipole algorithm in three dimensions, J. Comput. Phys., 155, 1999, 468-498.

[Dehnen02] W. Dehnen, A Hierarchical O(N) force calculation algorithm, J. Comput. Phys. 179, 2002, 27–42.

[Elliott02] W. Elliott and J. Board, Jr., Fast Fourier transform accelerated fast multipole algorithm, SIAM J. Sci. Computing, 17(2), 1996, 398-415.

[Epton95] M.A. Epton and B. Dembart, Multipole translation theory for the three-dimensional Laplace and Helmholtz equations, SIAM J. Sci. Comput., 16(4), 1995, 865-897.

[Greengard88] L. Greengard, The Rapid Evaluation Of Potential Fields in Particle Systems, MIT Press, Cambridge, 1988.

[Greengard87] L. Greengard and V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phys., 73, 1987, 325-348.

[GreengardL88] L. Greengard and V. Rokhlin, Rapid evaluation of potential fields in three dimensions, in Vortex Methods. Edited by C. Anderson and C. Greengard. Lecture Notes in Mathematics, Springer-Verlag, Berlin, 1360, 1988, 121-141.

[Greengard97] L. Greengard and V. Rokhlin, A new version of the fast multipole method for the Laplace equation in three dimensions, Acta Numerica, 6, 1997, 229-269.

[Gumerov2003] N.A. Gumerov, R. Duraiswami, and E.A. Borovikov, Data structure, optimal choice of parameters, and complexity results for generalized multilevel fast multipole methods in d dimensions, UMIACS TR 2003-28, University of Maryland, College Park, 2003.

[Gumerov2005] N.A. Gumerov and R. Duraiswami, Fast Multipole Methods for the Helmholtz Equation in Three Dimensions, Elsevier, Oxford, 2004.

[Hrycak98] T. Hrycak and V. Rokhlin, An improved fast multipole algorithm for potential fields, SIAM J.Sci. Comput., 19(6), 1998, 1804-1826.

[Lindsay01] K. Lindsay and R. Krasny, A particle method and adaptive treecode for vortex sheet motion in three-dimensional flow, J. Comput. Phys., 172, 2001, 879-907.

[Ong04] E.T. Ong, H.P. Lee, and K.M. Lim, A parallel fast Fourier transform on multipoles (FFTM) algorithm for electrostatics analysis of three-dimensional structures," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, 23, 2004, 1063-1072.

[Petersen94] H.G. Petersen, D. Soelvason, and J.W. Perram, The very fast multipole method, J. Chem. Phys. 101(10), 1994, 8870-8876.

[Rokhlin83] V. Rokhlin, Rapid solution of integral equations of classical potential theory, J. Comput. Phys., 60, 1983, 187-207.

[Tang03] Z. Tang, Fast Transforms Based On Structured Matrices with Applications to the Fast Multipole Method, Ph.D. thesis, University of Maryland, College Park, 2003.

[White96] C.A. White and M. Head-Gordon, Rotation around the quartic angular momentum barrier in fast multipole method calculations, J. Chem. Phys., 105(12), 1996, 5061-5067.

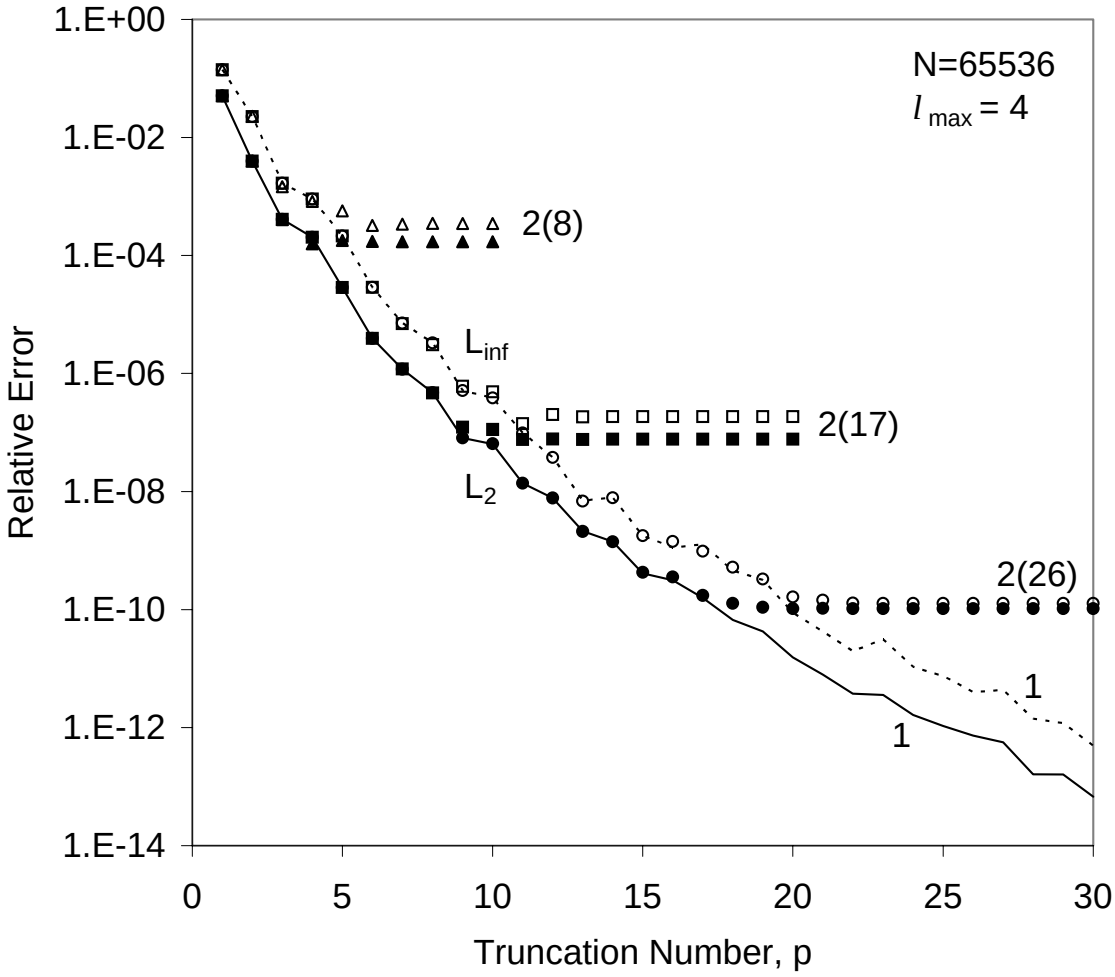

Figure 8: The relative errors ϵ_∞ (marked as L_{inf}) and ϵ_2 (marked as L_2) computed according Eq. (89) for the first 100 evaluation points ($M = 100$) as functions of the truncation number, p. The solid and dotted curves show computations using translation method #1, while the triangles, squares and discs correspond to computations using translation method #2 (exponential expansions). In the latter case the numbers in the parenthesis after 2 show the order of quadrature, $S(\epsilon)$, used for exponential representation. 65536 sources of equal intensity are randomly distributed inside a cube; $l_{\max} = 4$. Computations with double precision.

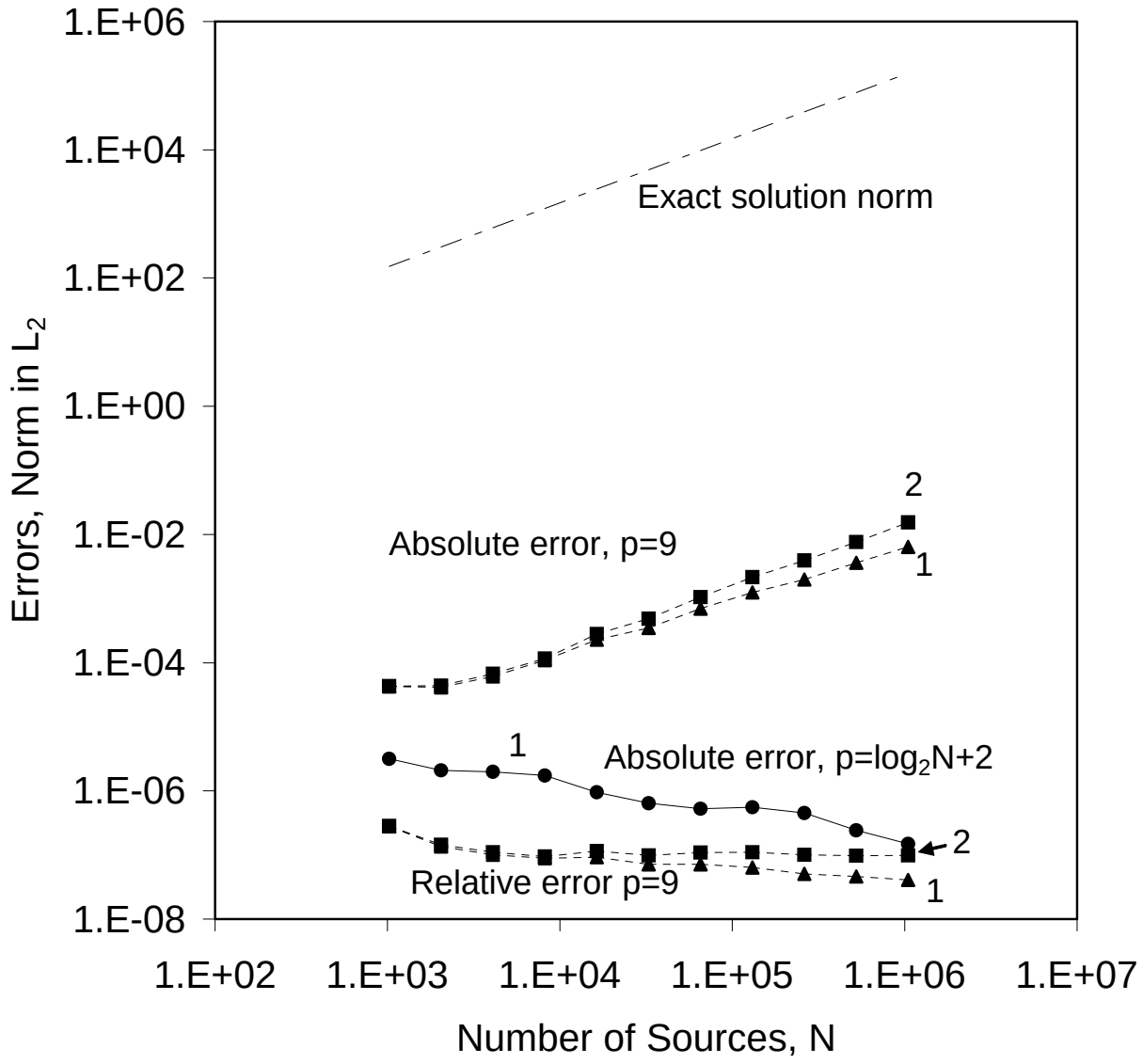

Figure 9: Dependences of the exact solution L_2 norm (the dash-dotted line) and absolute and relative errors in the L_2 norm (solid and dashed lines) on the size of the problem, N. The translation method used is indicated near the curves. In the Method #2 the quadrature of order 17 was used. The discs connected by the solid line show the absolute error for computations using p depending on N as shown. The squares and triangles connected by the dashed line show computations using constant $p = 9$.The sources of unit intensity are uniformly randomly distributed inside the unit cube. Error is measured based on the first 100 evaluation points.

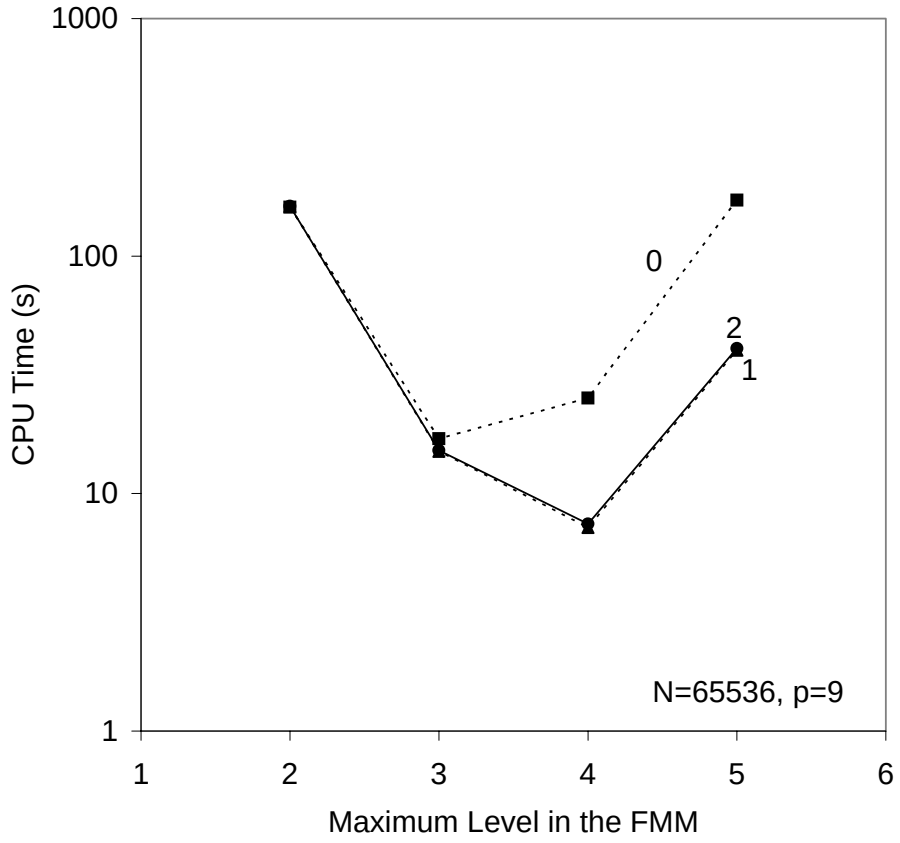

Figure 10: Dependence of the CPU time (Intel Xeon 3.2 GHz, 3.5 GB RAM) on the maximum level of space subdivision used in the FMM. Numbers near the curves indicate the translation method used (for Method #2, $S(\epsilon) = 17$). Sources of equal intensity are randomly uniformly distributed inside a cube.

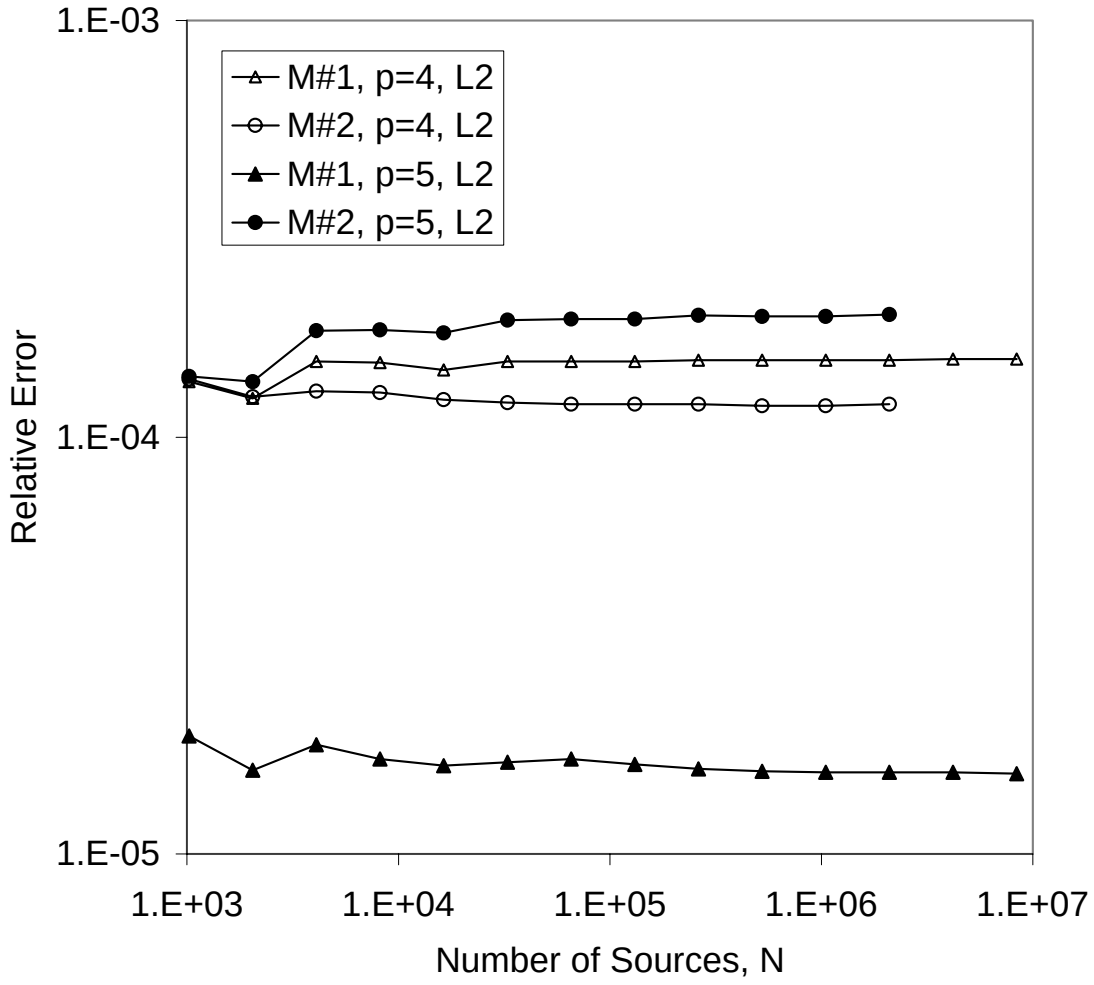

Figure 11: The relative errors ϵ_2 (data series are connected by the solid lines) as functions of the number of sources for computations with the FMM using different translation methods (the triangles for Method #1, and the discs for Method #2 with the 8th order quadrature) for truncation numbers $p = 4$ (the empty marks) and $p = 5$ (the filled marks). Error measurements are made over the first 100 evaluation points. The maximum level in the FMM is optimized for each N. Sources are distributed uniformly randomly inside a cube. Computations are made with double precision.

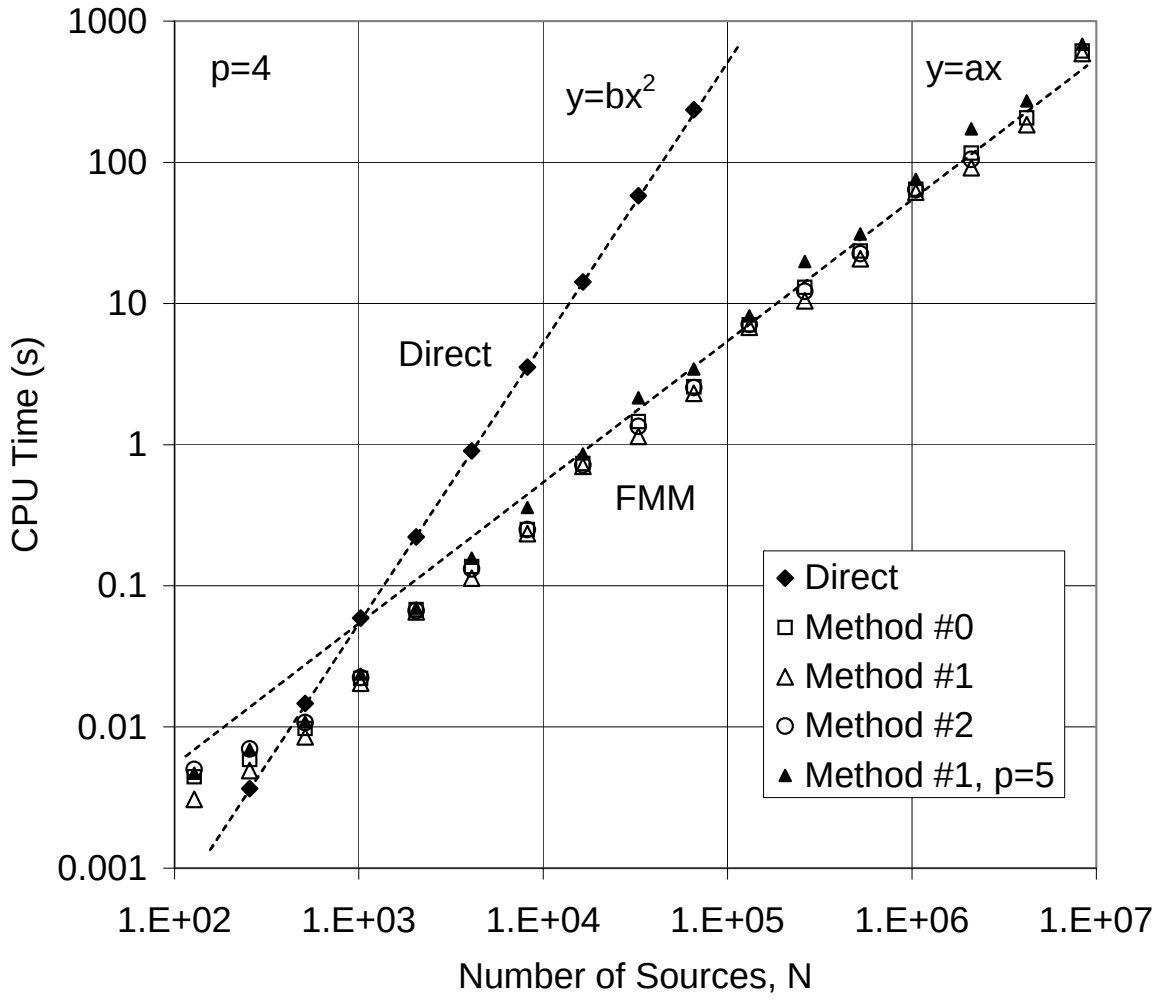

Figure 12: The CPU time in seconds required for computation of problems of size N using different methods. Data shown by filled rhombi corresponds to the direct solution, which complexity is scaled as $O(N^2)$ shown by the dashed line $y = bx^2$. FMM computations using Methods #0, #1, and #2 with truncation number $p = 4$ are shown by the empty squares, triangles, and discs, respectively. FMM computations using Method #1 and $p = 5$ are shown by smaller filled triangles. All the FMM computations are scaled asymptotically linearly at larger N. The dashed line $y = ax$ presents linear scaling. CPU is measured for the Intel Xeon 3.2 GHz processor, 3.5 GB RAM (double precision).

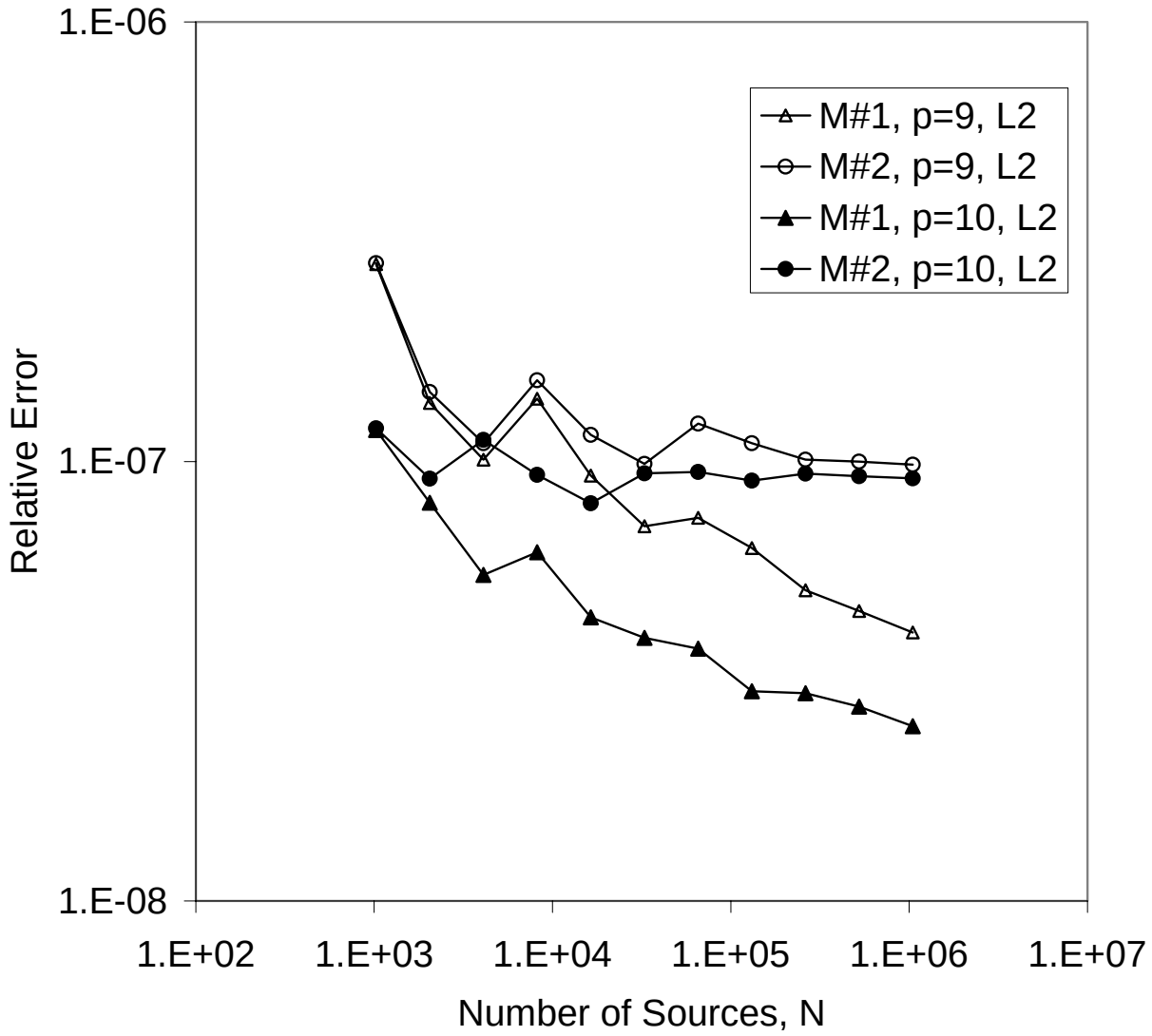

Figure 13: The same as Fig. 11 but for different truncation numbers and Method #2 using the 17th order quadrature.

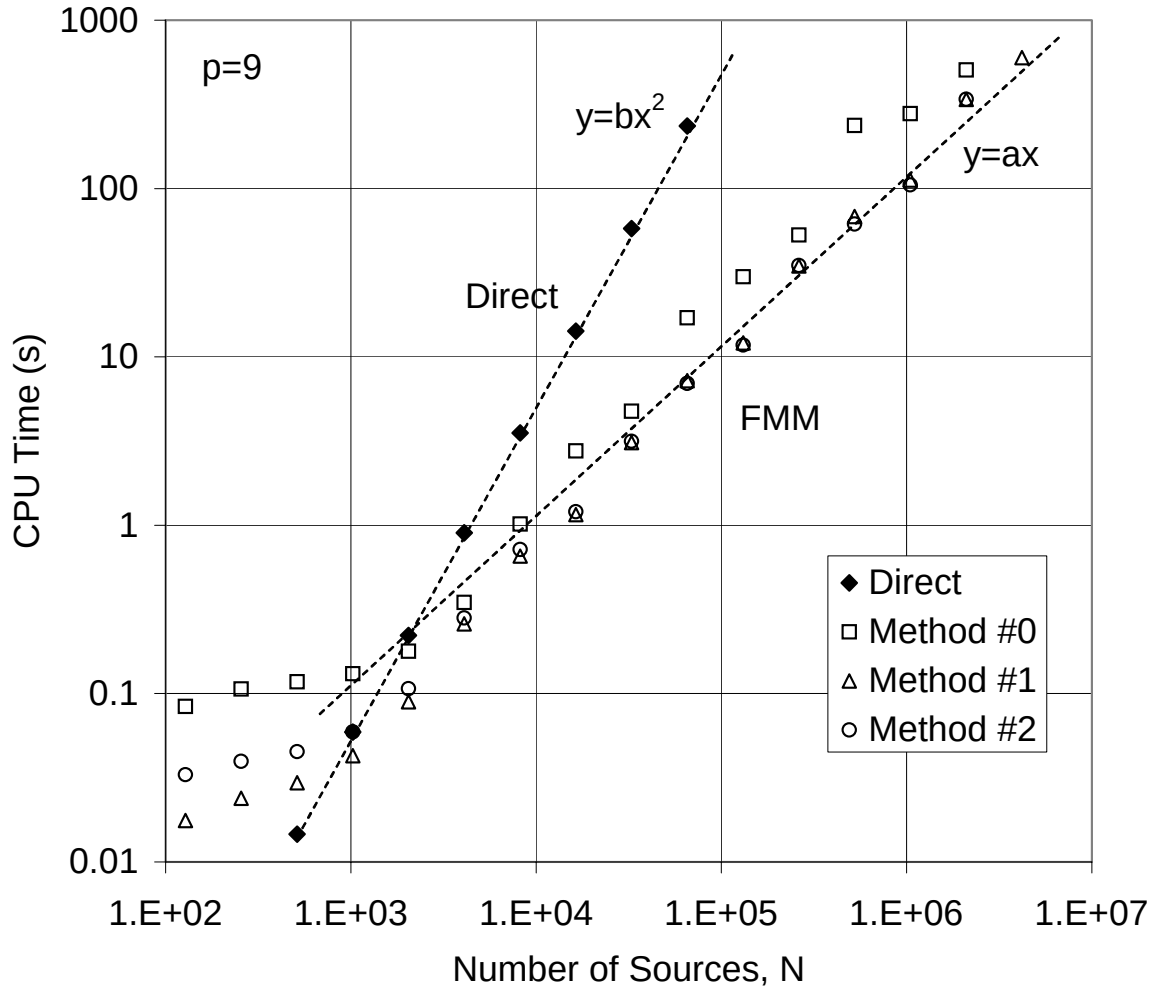

Figure 14: The same as Fig. 12, but for different p ($p = 9$) and for the 17th order quadrature for Method #2.

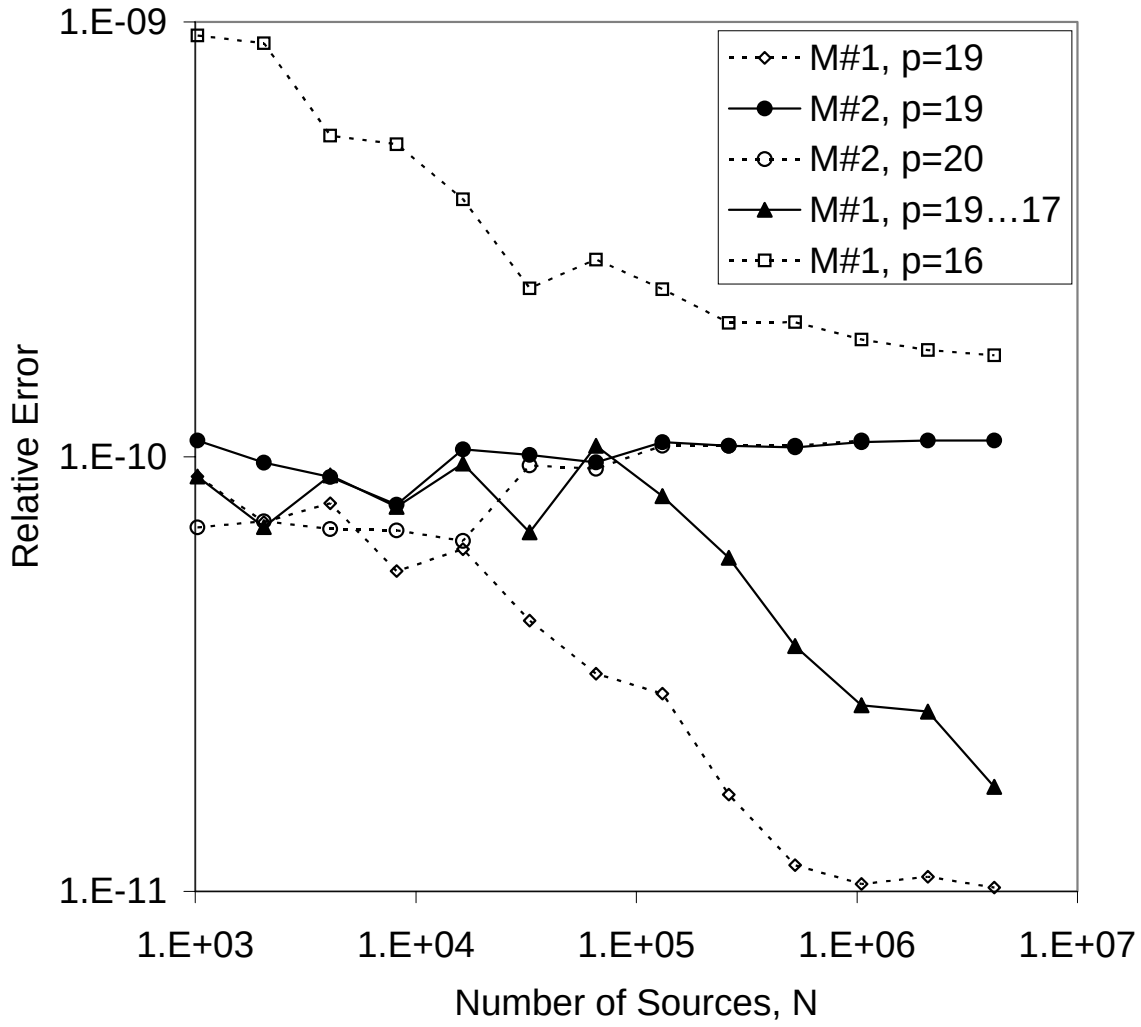

Figure 15: The same as Fig. 11 but for different truncation numbers and Method #2 using the 26th order quadrature. The curve marked by the dark triangles corresponds computed for variable p ($p = 19$ for $N = 2^{10}$; $p = 18$ for $N = 2^{10} - 2^{15}$; $p = 17$ for $N = 2^{16} - 2^{22}$).

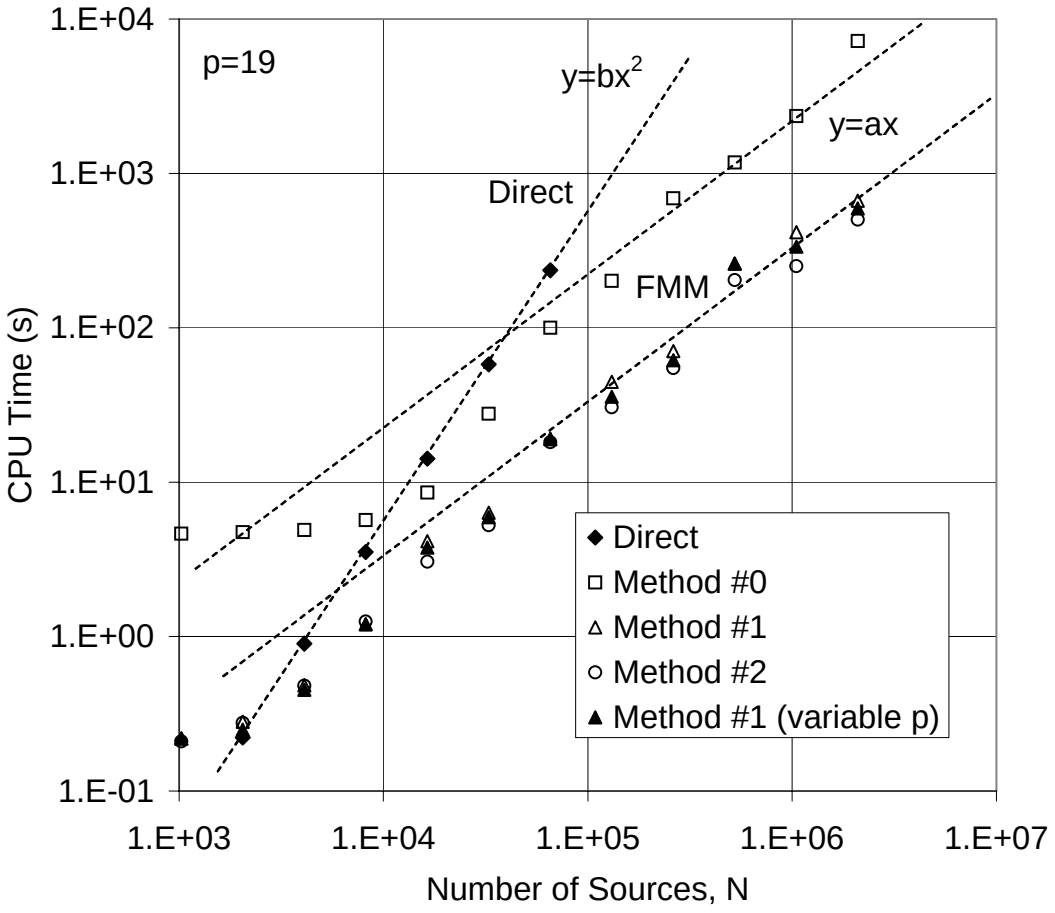

Figure 16: The same as Fig. 12, but for different p ($p = 19$) and for the 26th order quadrature for Method #2. The dark triangles show performance of Method #1 where p can be reduced for particular N according the error in the L_2 norm (see Fig. 19).

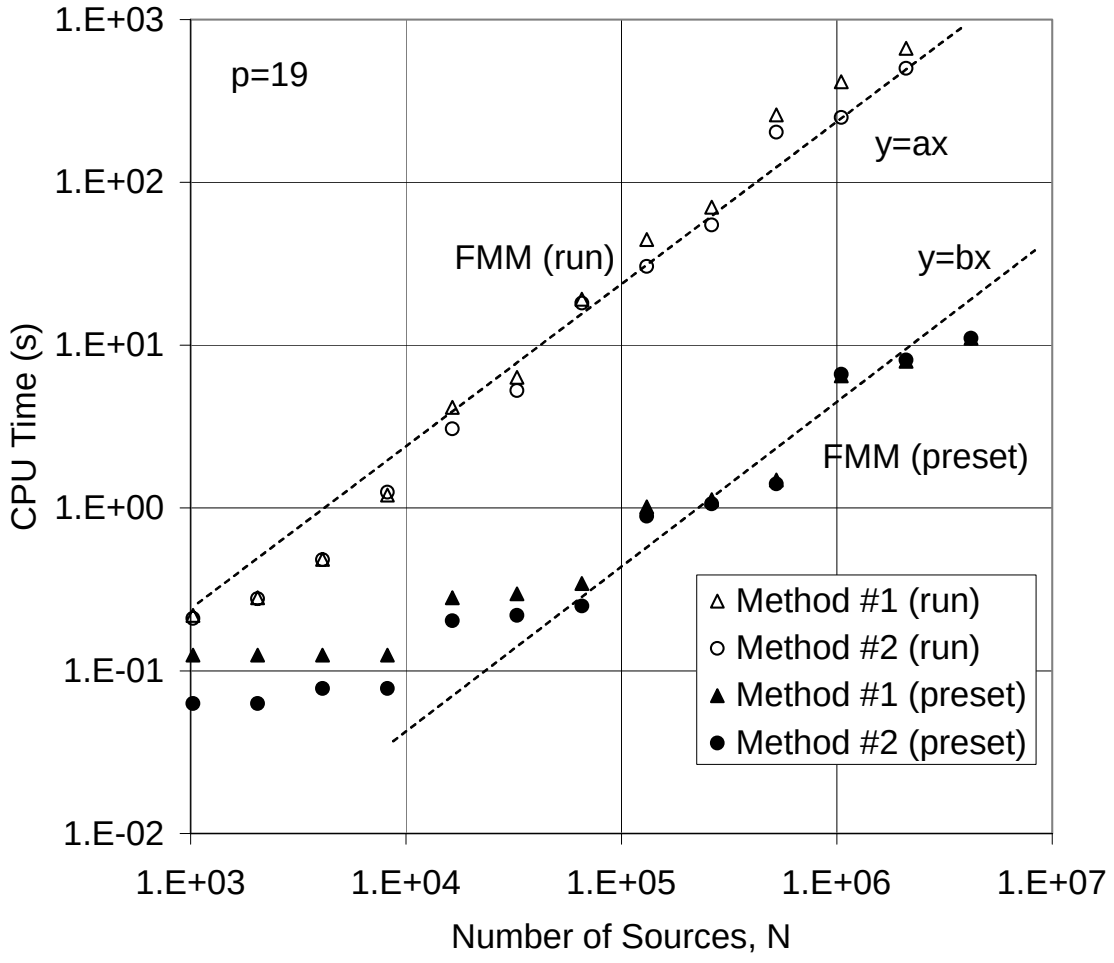

Figure 17: Depenedences of the CPU time for the preset and the run step of the FMM. $p = 19$, $S(\epsilon) = 26$.