pfft++ – A general and extensible fast integral equation solver based on a pre-corrected FFT algorithm [*]

Zhenhai Zhu , Ben Song and Jacob White

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology, Cambridge, MA 02139
{zhzhu, bsong01, white}@mit.edu

Abstract

This paper describes an efficient algorithm to solve integral equations with complicated kernels. The algorithm can handle both piece-wise constant and high-order elements in the collocation method and the Galerkin's method. A general and extensible fast integral equation solver has been developed using the c++ generic programming technique. Numerical experiments show that the memory usage and CPU time of the fast solver is nearly $O(N)$ for various kernels.

1. Introduction

Integral equation methods have been used in applications as diverse as parasitic parameter extraction of integrated circuit interconnects and packages [1, 2], antenna characterization and radar cross section calculation [3, 4], computation of the molecular electric potential [5, 6], computational aerodynamics [7, 8], computational fluid dynamics [9].

Even though numerous fast algorithms already exist for efficiently solving the integral equations, such as Fast Multipole Method (FMM) [10, 11, 12, 13], hirarchical SVD [14], panel clustering method [15] and the pre-corrected FFT (pFFT) algorithm [16], the practical implementation of such methods may still seem daunting to researchers and engineers, who are most often not specialists in fast integral equation solvers. As a result many existing codes still use the traditional dense matrix approaches, which need $O(N^2)$ memory and at least $O(N^2)$ CPU time. One of the objects of this work is to provide a flexible and extensible code to the public domain so that the researchers can easily accelerate their codes. Hence we want to use an algorithm that is flexible enough to handle the integral kernels commonly used in the above mentioned engineering applications.

Though not as good as FMM's more than ten digit accuracy, pFFT's four to five digit accuracy is good enough for most engineering applications, where the accuracy requirement is usually modest. More importantly, the pFFT method is almost kernel-independent. For example, it could easily handle both Helmholtz kernel and Laplace kernel and their close relatives in a unified framework. This makes it a particularly good algorithm for our fast solver.

2. Mathematical Preliminaries

The integral equation method is a well studied subject [17, 15]. It is well-known that a large class of linear partial differential equations with appropriate boundary conditions could be casted into equivalent integral equations [18, 19, 20]. A general form for these integral equations is

$$\int_S dS' K(\vec{r'},\vec{r})\rho(\vec{r'}) = f(\vec{r}),\ \ \vec{r}\in S \tag{1}$$

where $f(\vec{r})$ is a known function, usually related to the known right hand side term in the original partial differential equation, and $K(\vec{r'},\vec{r})$ could be the commonly used single-layer kernel, double-layer kernel or other more complicated kernels. For mixed boundary condition, we might even have two kernels in equation (1). But this will not change the nature of the problem and the method in this paper. So we focus on the single kernel integral equation only. An abstract form of this kernel is

$$K(\vec{r'},\vec{r}) = \mathcal{F}_1(\mathcal{F}_2(G(\vec{r'},\vec{r}))) \tag{2}$$

where $G(\vec{r'},\vec{r})$ is the Green's function for the partial differential equation, and the possible options for operator $\mathcal{F}_1(\cdot)$ and $\mathcal{F}_2(\cdot)$ are

$$\mathcal{F}_1(\cdot) = U(\cdot), \frac{d(\cdot)}{dx(\vec{r})}, \frac{d(\cdot)}{dy(\vec{r})}, \frac{d(\cdot)}{dz(\vec{r})}, \frac{d(\cdot)}{dn(\vec{r})}, \tag{3}$$

and

$$\mathcal{F}_2(\cdot) = U(\cdot), \frac{d(\cdot)}{dx(\vec{r'})}, \frac{d(\cdot)}{dy(\vec{r'})}, \frac{d(\cdot)}{dz(\vec{r'})}, \frac{d(\cdot)}{dn(\vec{r'})}, \tag{4}$$

and $U(\cdot)$ is the identity operator.

The standard procedure to solve equation (1) numerically is to discretize it by means of projection [15] and solve the resultant linear system with an iterative method [21, 22], such as GMRES [23]. Let X be the infinite-dimensional functional space in which the exact solution of equation (1) lies, and assume that $B_n \subset X$ and $T_n \subset X$ are its subspaces with spans $\{b_j(\vec{r}), j = 1,2,...,n\}$ and $\{t_i(\vec{r}), i = 1,2,...,n\}$, where n is the dimension of both subspaces. In general, the solution of the equation (1) is not in subspace B_n. Therefore, the approximate solution

$$\rho_n(\vec{r}) = \sum_{j=1}^{n} \alpha_j b_j(\vec{r}) \in B_n \tag{5}$$

generates an error

$$e_n(\vec{r}) = \int_S dS' K(\vec{r'},\vec{r})\rho_n(\vec{r'}) - f(\vec{r}) = \phi(\vec{r}) - f(\vec{r}),\ \ \vec{r}\in S \tag{6}$$

and the unknown expansion coefficients α_i could be computed by enforcing the projection of the error into T_n to vanish, i.e.,

$$< t_i(\vec{r}), e_n(\vec{r}) >=< t_i(\vec{r}), \phi(\vec{r}) > - < t_i(\vec{r}), f(\vec{r}) >= 0,\ \ i = 1,2,...,n \tag{7}$$

or

$$\sum_{j=1}^{n} \alpha_j \int_{\Delta_i^t} dS t_i(\vec{r}) \int_{\Delta_j^b} dS' K(\vec{r'},\vec{r}) b_j(\vec{r'}) = \int_{\Delta_i^t} dS t_i(\vec{r}) f(\vec{r}),\ \ i = 1,2,...,n, \tag{8}$$

[*]This work was sponsored by the Semiconductor Research Corporation, the NSF program in computer-aided design, and grants from Synopsys, Compaq and Intel.

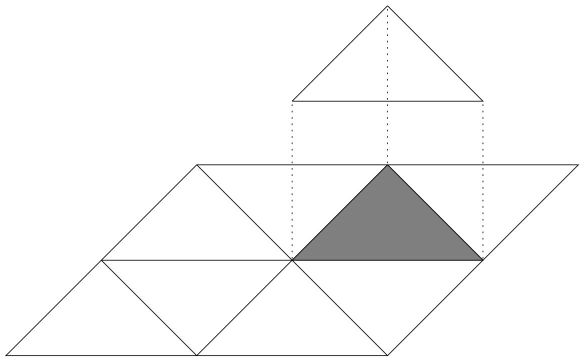

Figure 1: A piece-wise constant basis function, shaded area is its support

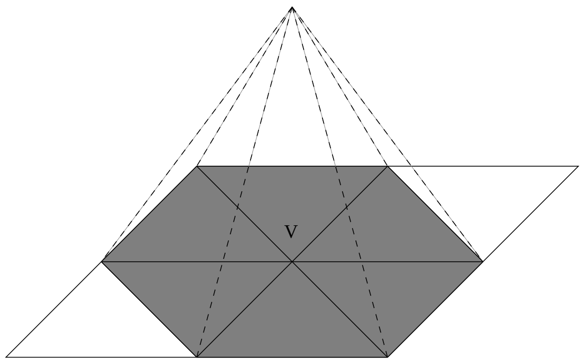

Figure 2: A piece-wise linear basis function associated with the vertex V, where the shaded area is its support

where Δ_i^t and Δ_j^b are the support of the basis function $t_i(\vec{r})$ and $b_j(\vec{r})$, respectively. In matrix, equation (8) becomes

$$[A]\bar{\alpha} = \bar{f} \tag{9}$$

where

$$A_{i,j} = \int_{\Delta_i^t} dS t_i(\vec{r}) \int_{\Delta_j^b} dS' K(\vec{r'}, \vec{r}) b_j(\vec{r'}) \tag{10}$$

The commonly used basis functions in B_n or T_n are low-order polynomials with local support [15]. Figure 1 shows a piece-wise constant basis function whose support is a panel. Figure 2 shows a vertex-based piece-wise linear basis function whose support is the union of a cluster of panels sharing the vertex with which the basis function is associated.

When the ith testing function is $t_i(\vec{r}) = \delta(\vec{r} - \vec{r_{c,i}})$, where $\vec{r}_{c,i}$ is the collocation point, the discretization method is called the collocation method. And when $B_n = T_n$, the discretization method is called the Galerkin's method.

3. Philosophical Preliminaries

Since forming matrix A and computing the matrix vector product in (9) all require $O(N^2)$ arithmetic operations, it is obvious that using an iterative method to solve equation (9) needs at least $O(N^2)$ time, where N is the size of the matrix A. This could be very expensive for large N. Many fast algorithms avoid forming matrix A explicitly and compute the matrix vector product approximately, which only needs $O(N)$ or $O(Nlog(N))$ operations [11, 24, 25].

The Pre-corrected FFT (pFFT) algorithm was originally proposed in [16, 25], where the detailed steps to accelerate a single-layer integral operator were shown. The basic idea of pFFT is to separate the potential computation into far-field part and near-field part. The far-field potential is computed by using the grid charges on a uniform 3D grid to represent charges on the panels. The near-field potential is compute directly. The algorithm has four steps: Projection, Convolution, Interpolation and Nearby interaction. The effect of this algorithm is to replace the matrix vector product $A\bar{\alpha}$ in equation (9) with $(D+IHP)\bar{\alpha}$, where D is the direct matrix that represents the nearby interaction, I is the interpolation matrix, H is the convolution matrix, and P is the projection matrix. Matrices D, I and P are sparse, hence their memory usage is $O(N_p)$, where N_p is the number of panels, and their product with a vector needs only $O(N_p)$ work. The matrix H is a multilevel Toeplitz matrix. Hence its memory usage is $O(N_g)$ and its product with a vector could be computed by using FFT in $O(N_g log(N_g))$ operations [26], where N_g is the number of grid points. Therefore, the overall computational complexity of $(D+IHP)\bar{\alpha}$ is $O(N_p) + O(N_g log(N_g))$. For some problems, usually small or medium sized ones, N_g might be larger. Hence the computational complexity is $O(N_g log(N_g))$. For other problems, usually large-sized ones, the computational complexity is nearly $O(N_p)$.

Unlike [16, 25], we use polynomials in both interpolation and projection steps. Hence the interpolation matrix I and projection matrix P are completely independent of the Green's function $G(\vec{r}, \vec{r'})$ in equation (2). This makes it much easier to handle the complicated kernels $K(\vec{r'}, \vec{r})$ in (2). It also makes it straight forward to treat piecewise constant basis and high-order basis in either collocation or Galerkin's method in a unified framework. This is particularly important from implementation point of view.

4. Pre-corrected FFT algorithm

In this section, we will use a simple 2D example to show how to generate the four matrices, $[I]$, $[P]$, $[H]$ and $[D]$. Generalization of the procedure to the 3D cases is straight forward. The algorithm presented here is general enough such that the general integral operator in equation (1) discretized either by the collocation method or by the Galerkin's method using either piece-wise constant element or high-order element could be handled in a unified framework.

4.1 Interpolation matrix

We start with the interpolation, the third and the easiest step in the four-step pFFT algorithm.

Suppose the potential on the uniform grids has been computed through the first two steps, namely the projection and the convolution, we could use a simple polynomial interpolation scheme to compute the potential at any point within the region covered by the grids. Figure 3 shows a 2D 3×3 uniform grid (called interpolation stencil in this paper), more points could be used to get more accurate results. The triangle inside the grid represents the local support Δ_i^t in equation (8). The simplest set of polynomial functions for the interpolation is $f_k(x,y) = x^i y^j, i, j = 0, 1, 2, k = 2i + j$. The potential at any point can be written as a linear combination of these polynomials,

$$\phi(x,y) = \sum_k c_k f_k(x,y) = \bar{f}^t(x,y)\bar{c} \tag{11}$$

where $\bar{c}$ is a column vector and t stands for transpose. Matching $\phi(x,y)$ in (11) with the given potential at each grid point results in a set of linear equations. In matrix form, it is

$$[F]\bar{c} = \bar{\phi}_g \tag{12}$$

where the j-th row of the matrix $[F]$ is the set of polynomials $\bar{f}(x,y)$ evaluated at the jth grid point (x_j, y_j), and $\phi_{g,j}$ is the given potential at point (x_j, y_j). Solving for $\bar{c}$ and substituting it back into (11) yields

$$\phi(\vec{r}) = \phi(x,y) = \bar{f}^t(x,y)[F]^{-1}\bar{\phi}_g = \bar{D}_0^t(\vec{r})\bar{\phi}_g \tag{13}$$

It should be noted that matrix $[F]$ in (12) is only related to the distance between points in the uniform grid and the specific set of interpolation polynomials chosen in the algorithm. So the inverse of matrix $[F]$ is done only once. And since the size of the matrix is rather small (9×9 in this simple 2D case), computing its inverse is inexpensive. It is possible that the number of polynomials is not equal to the number of points in the interpolation grid. In this case the inverse becomes psuedo inverse, which is computed using the singular value decomposition (SVD) [22].

It easily follows that the derivative of the potential at a point $\bar{r}$ with respect to α is

$$\frac{d\phi(\bar{r})}{d\alpha} = \frac{d}{d\alpha}\bar{f}^t(\bar{r})[F]^{-1}\bar{\phi}_g = \bar{D}^t_\alpha(\bar{r})\bar{\phi}_g \tag{14}$$

where α stands for x or y. Hence the gradient of the potential at $\bar{r}$ is

$$\nabla\phi(\bar{r}) = (\hat{x}\bar{D}^t_x(\bar{r}) + \hat{y}\bar{D}^t_y(\bar{r}))\bar{\phi}_g \tag{15}$$

and the normal derivative of the potential at point $\bar{r}$ is

$$\frac{d\phi(\bar{r})}{dn} = \hat{n}\cdot\nabla\phi(\bar{r}) = (n_x\frac{d\bar{f}^t(\bar{r})}{dx} + n_y\frac{d\bar{f}^t(\bar{r})}{dy})[F]^{-1}\bar{\phi}_g = \bar{D}^t_n(\bar{r})\bar{\phi}_g \tag{16}$$

where n_x and n_y are the projection of the unit normal vector of the function support Δ^t_i along x and y direction. Using the notation in (3), equations (13), (14) and (16) could be written as

$$\mathcal{F}_1(\phi(\bar{r})) = \bar{D}^t_\beta(\bar{r})\bar{\phi}_g \tag{17}$$

where $\bar{D}^t_\beta(\bar{r})$ stands for $\bar{D}^t_0(\bar{r})$, $\bar{D}^t_x(\bar{r})$, $\bar{D}^t_y(\bar{r})$ or $\bar{D}^t_n(\bar{r})$.

As described in section **??**, we want to compute

$$\Psi_i = \int_{\Delta^t_i} dS\mathcal{F}_1(\phi(\vec{r}))t_i(\vec{r}),\ \ i = 1,2,..,N_t. \tag{18}$$

where N_t is the number of testing basis functions. Substituting (17) into (18) yields

$$\Psi_i = \int_{\Delta^t_i} dSt_i(\vec{r})\bar{D}^t_\beta(\bar{r})\bar{\phi}_g = (\bar{W}^{(i)}_\beta)^t\bar{\phi}_g,\ \ i = 1,2,..,N_t, \tag{19}$$

where $\bar{W}^{(i)}_\beta$ stands for $\bar{W}^{(i)}_0$, $\bar{W}^{(i)}_x$, $\bar{W}^{(i)}_y$ and $\bar{W}^{(i)}_n$. If the collocation method is used, then $\bar{W}^{(i)}_\beta$ in equation (19) could be simplified as

$$\bar{W}^{(i)}_\beta = \bar{D}_\beta(x_c,y_c),\ \ i = 1,2,..,N_t, \tag{20}$$

where (x_c, y_c) is the collocation point.

When the piece-wise constant testing function is used, the support Δ^t_i is the panel associated with it, as shown in figure 1. When the linear testing function is used, Δ^t_i is a cluster of panels, as shown in figure 2. Apparently, computing elements of $\bar{W}^{(i)}_\beta$ for higher order basis functions could be more expensive because integrating over a cluster of panels needs more quadrature points than integrating over a single panel.

In matrix format, equation (19) becomes

$$\bar{\Psi} = [I]\bar{\phi}_g \tag{21}$$

where $[I]$ is an $N_t \times N_g$ matrix, and N_g is the number of grid points. To cover the local support of a basis function, only a small number of the interpolation grid points are needed, as shown in figure 3. More importantly, the potential ϕ in (6) is a smooth function of $\vec{r}$ when $|\vec{r} - \vec{r}'|$ is large. Hence low-order polynomials in (11) are sufficient to well approximate the distant interaction. Therefore, computing each Ψ_i through interpolation only involves grid potentials at a few grid points. This implies that each row of the interpolation matrix $[I]$ is rather sparse. The non-zero elements in the i-th row of the matrix $[I]$ are just the elements

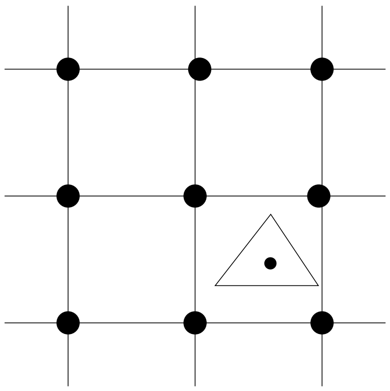

Figure 3: 2-D pictorial representation of the interpolation step, the interpolation stencil size is 3

of the row vector $(\bar{W}^{(i)}_\beta)^t$ in (19) or (20). And the number of nonzeros's is equal to the interpolation stencil size.

4.2 Projection matrix

Figure 4 shows a 2D pictorial representation of the projection step. Similar to the previous section, a triangle is used to represent the support of a basis function. A 3×3 projection grid (called projection stencil in this paper) is assumed here and obviously more points could be used if higher accuracy is desired.

We start with a point charge ρ_p at point **S** on the triangle, shown in figure 4. The potential at point **E** due to this point charge is

$$\phi^{(1)}_E = \rho_p G(\vec{r}_s, \vec{r}_E). \tag{22}$$

The purpose of the projection is to find a set of grid charges $\bar{\rho}_g$ on the projection grid points such that they generate the same potential at point **E**, i.e.,

$$\phi^{(2)}_E = \sum_i \rho_{g,i} G(\vec{r}_i, \vec{r}_E) = (\bar{\rho}_g)^t\bar{\phi}_g = \phi^{(1)}_E \tag{23}$$

where $\phi_{g,i} = G(\vec{r}_i, \vec{r}_E)$. We could use the same set of polynomials in (11) to expand the Green's function

$$G(\vec{r}, \vec{r}_E) = \sum_k f_k(\vec{r})c_k = \bar{f}^t(\vec{r})\bar{c}. \tag{24}$$

Matching both sides at each grid point $\vec{r}_i$ yields a linear system

$$[F]\bar{c} = \bar{\phi}_g, \tag{25}$$

where F is same as that in (12). Substituting the solution $\bar{c} = F^{-1}\bar{\phi}_g$ into (24) and evaluating it at point **S** yields

$$G(\vec{r}_s, \vec{r}_E) = \bar{f}^t(\vec{r}_s)F^{-1}\bar{\phi}_g. \tag{26}$$

In light of (22) and (23) we have

$$(\bar{\rho}_g)^t = \rho_p\bar{f}^t(\vec{r}_s)F^{-1}, \tag{27}$$

the projection charges for a point charge.

A charge distribution $b_j(\vec{r})$ on the jth basis function support could be regarded as a linear combination of an infinite number of point charges. Equation (27) implies that the projection charges are linearly proportional to the point charge, hence it easily follows that the projection charges for the charge distribution $b_j(\vec{r})$ is

$$(\bar{\rho}^{(j)}_g)^t = \left(\int_{\Delta^b_j} dSb_j(\vec{r})\bar{f}^t(\vec{r})\right)[F]^{-1}. \tag{28}$$

If the piece-wise constant basis function is used, equation (28) becomes

$$\left(\bar{\rho}_g^{(j)}\right)^t = \left(\int_{\Delta_j^b} dS \bar{f}^t(\vec{r})\right)[F]^{-1}. \tag{29}$$

We usually have to use more than one basis function in the approximate solution, as implied by equation (5). In this case, the total charge on each grid point is the accumulation of the grid charge due to each basis function. Assuming there are N_b basis functions and N_g grid points, the relation between the total grid charges $\bar{Q}_g$ and the magnitude of basis functions $\bar{\alpha}$ in (5) is

$$\bar{Q}_g = \sum_{j=1}^{N_b} \alpha_j \bar{\rho}_g^{(j)} = [P]\bar{\alpha}, \tag{30}$$

where $[P]$ is an $N_g \times N_b$ matrix. Due to the locality of the basis support, the projection grid for each basis function has only a small number of points. And similar to interpolation matrix, the Green's function in (24) is a smooth function of $\vec{r}$ when $|\vec{r}-\vec{r}_E|$ is large. Hence low-order polynomials in (24) are sufficient to approximate it well. This implies that each column of the projection matrix $[P]$ is rather sparse. The non-zero elements in the j-th column of matrix $[P]$ are the elements of the column vector $\bar{\rho}_g^{(j)}$ in equation (28) or (29). And the number of nonzero's is equal to the projection stencil size.

If the kernel has a differential operator inside the integral, the potential at point **E** due to a point charge is

$$\phi_E^{(1)} = \frac{\partial}{\partial\beta(\vec{r}_s)}[\rho_p G(\vec{r}_s,\vec{r}_E)] = \frac{\partial}{\partial\beta(\vec{r}_s)}[\rho_p \bar{f}^t(\vec{r}_s)F^{-1}\bar{\phi}_g]. \tag{31}$$

where β stands for x, y or n. We again want to find a set of grid charges $\bar{\sigma}_\beta$ on the projection grid points such that they generate the same potential at point **E**, i.e.,

$$\phi_E^{(2)} = \sum_i \sigma_{\beta,i} G(\vec{r}_i,\vec{r}_E) = (\bar{\sigma}_\beta)^t \bar{\phi}_g = \phi_E^{(1)}. \tag{32}$$

Equations (31) and (32) imply that the projection charges are

$$(\bar{\sigma}_\beta)^t = \frac{\partial}{\partial\beta(\vec{r}_s)}\left(\rho_p \bar{f}^t(\vec{r}_s)F^{-1}\right). \tag{33}$$

Similar to the single-layer operator case, the projection charges for a charge distribution $b_j(\vec{r})$ on the jth basis function support is

$$\left(\bar{\sigma}_\beta^{(j)}\right)^t = \left(\int_{\Delta_j^b} dS b_j(\vec{r})\frac{\partial}{\partial\beta(\vec{r})}\bar{f}^t(\vec{r})\right)[F]^{-1}. \tag{34}$$

The projection matrix for the kernel with a differential operator is structurely identical to the matrix $[P]$ in equation (30). The non-zero elements in the j-th column of the matrix are the elements of the column vector $\bar{\sigma}_\beta^{(j)}$ in equation (34).

4.3 Convolution matrix and fast convolution by FFT

By definition, the relation between the grid potential $\bar{\phi}_g$ in (21) and grid charge $\bar{Q}_g$ in (30) is

$$\phi_{g,j} = \sum_i G(\vec{r'}_i,\vec{r}_j)Q_{g,i} \tag{35}$$

In matrix form, it is

$$\bar{\phi}_g = [H]\bar{Q}_g \tag{36}$$

where the matrix H is the so-call convolution matrix. Since the Green's function is position invariant and $\bar{\phi}_g$ and $\bar{Q}_g$ are defined on the same set

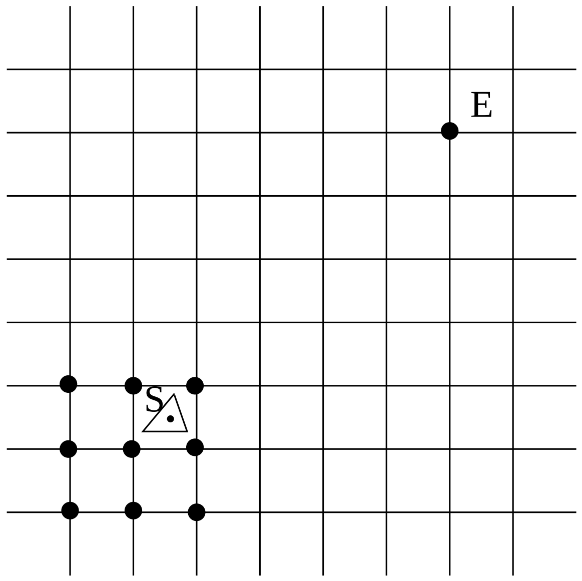

Figure 4: 2-D pictorial representation of the projection step, the projection stencil size is 3

of uniform grid, we have

$$H_{i,j} = G(\vec{r'}_i,\vec{r}_j) = G(\vec{r}_i,\vec{r}_j) = G(\vec{r}_i-\vec{r}_j,0). \tag{37}$$

Matrix H is a multilevel Toeplitz matrix [26]. The number of levels is 2 and 3 for 2D cases and 3D cases, respectively. It is well-known that the storage of a Toeplitz matrix only needs $O(N)$ memory and a Toeplitz matrix vector product can be computed in $O(Nlog(N))$ operations using FFT [26], where N is the total number of grid points. It should be pointed out that convolution matrix H being a Toeplitz matrix is hinged upon the position invariance of the Green's function. Fortunately most commonly used Green's functions are position invariant.

4.4 Direct matrix and pre-correction

Substituting equation (36) and (30) into (21) yields

$$\bar{\Psi} = [I][H][P]\bar{\alpha} \tag{38}$$

In view of (18), (7) and (9), this implies

$$A = [I][H][P]. \tag{39}$$

As pointed out in previous three sections, the sparse representation of matrix A in (39) reduces the memory usage and computing time for matrix vector product dramatically. Unfortunately, the calculations of the potential on the grid using (39) do not accurately approximate the nearby interaction. It is proposed in [25] that the nearby interaction should be computed directly and the inaccurate contributions from the use of grid should be removed. Figure 5 shows how the nearby neighboring basis supports are defined. The empty circle in middle of the solid dots are the center of the so-called direct stencil and the stencil size in figure 5 is 5. The shaded triangle represents the source, and the other empty triangles represent the targets where Ψ in equation (18) is to be evaluated. Only those triangles within the region covered by the direct stencil are considered to be nearby neighbors to the source. And the direct interaction between this list of nearby neighbors and the source is just $A_{i,j}$ defined in (10), where i is the index of the shaded triangle representing the source and $j \in \mathcal{N}_i$, the nearby neighbor set for the ith source. The pre-corrected direct matrix element is

$$D_{i,j} = A_{i,j} - (\bar{W}_\beta^{(i)})^t[H_L]\bar{\rho}_g^{(j)}, \quad j \in \mathcal{N}_i \tag{40}$$

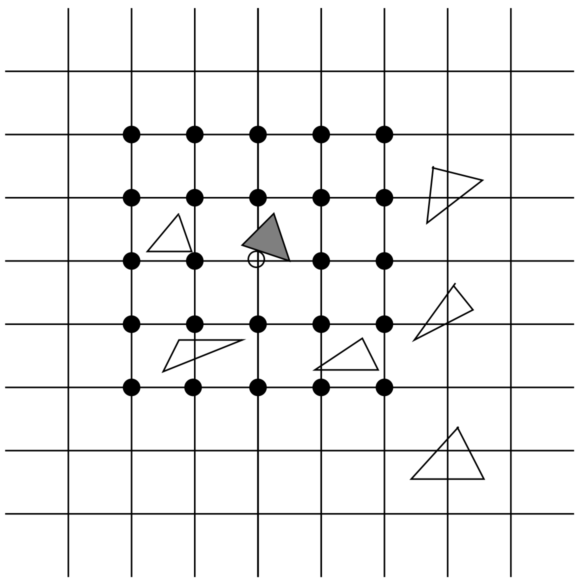

Figure 5: 2-D pictorial representation of the nearby interaction. Direct stencil size is 5.

Table 1: Relation between operator pair and the interpolation matrix and the projection matrix

$\mathcal{F}_1$	$U(\cdot)$	$\frac{d(\cdot)}{dx'}, \frac{d(\cdot)}{dy'}$	$\frac{d(\cdot)}{dn'}$
interpolation	$\overline{W}_0^{(i)}$ in (19)	$\overline{W}_x^{(i)}, \overline{W}_y^{(i)}$ in (19)	$\overline{W}_n^{(i)}$ in (19)
$\mathcal{F}_2$	$U(\cdot)$	$\frac{d(\cdot)}{dx}, \frac{d(\cdot)}{dy}$	$\frac{d(\cdot)}{dn}$
projection	$\overline{\rho}_g^{(j)}$ in (28)	$\overline{\sigma}_x^{(j)}, \overline{\sigma}_y^{(j)}$ in (34)	$\overline{\sigma}_n^{(j)}$ in (34)

where $(\overline{W}_\beta^{(i)})^t$ is defined in equation (19), $\overline{\rho}_g^{(j)}$ is defined in equation (28) and (34), and $[H_L]$ is a small convolution matrix (not to be confused with $[H]$ in (39)) that relates the potential on the grid points around basis support Δ_i^t and the charge on the grid points around basis support Δ_j^b. It is intuitive from figure 5 that $\mathcal{N}_i$ is a very small set. Hence the direct matrix D is very sparse and the sparsity of D is dependent upon the size of the direct stencil. Larger stencil size means more neighboring triangles in figure 5 and hence more computation in (40).

Since matrix $[H_L]$ in (40) is rather small, the FFT does not speed up the computation much. However, there are other ways to reduce the operation count. Because the grid is uniform and the Green's function is position invariant, only a few matrices $[H_L]$ are unique. So we could pre-compute them once and use them to pre-correct all the nearby interactions in the direct matrix $[D]$.

4.5 A summary of the four matrices

In view of (38), (39) and (40), the matrix vector product is computed efficiently using

$$[A]\overline{\alpha} = ([D] + [I][H][P])\overline{\alpha}. \qquad (41)$$

Sections 4.1 and 4.2 are summarized in table 1. It is clear by now that the interpolation matrix $[I]$ and the projection matrix $[P]$ are independent of the Green's function. Matrix $[I]$ is only related to the operator $\mathcal{F}_1$ and the testing functions. And matrix $[P]$ is only related to the operator $\mathcal{F}_2$ and the basis functions.

The direct matrix, however, is dependent upon all the above information. So we have to set up one direct matrix for each $\mathcal{F}_1$ and $\mathcal{F}_2$ operator pair. The convolution matrix, on the other hand, is only related to the Green's function and the location of grid points. It is not related to $\mathcal{F}_1$ or $\mathcal{F}_2$. So we only need to set up one convolution matrix for each unique Green's function.

In addition, if the Galerkin's method is used, the basis function $b_j(\vec{r})$ in equation (28) or (34) is identical to the testing function $t_i(\vec{r})$ in equation (19). It is easy to check that $\overline{W}_0^{(i)} = \overline{\rho}_g^{(j)}$, $\overline{W}_x^{(i)} = \overline{\sigma}_x^{(j)}$, $\overline{W}_y^{(i)} = \overline{\sigma}_y^{(j)}$ and $\overline{W}_n^{(i)} = \overline{\sigma}_n^{(j)}$. This implies a duality relation

$$[I] = [P]^t. \qquad (42)$$

4.6 Implementation

Base upon the algorithm described above, we have developed a C++ program called pfft++, using the generic programming technique [27, 28, 29]. The whole algorithm includes two major parts: forming the four matrices I, P, D and H, and computing the matrix vector product using (41). Since the matrices I and P are not related to the Green's function, they are formed separately so that they could be used for different Green's functions. This is particularly useful when for example a Helmholtz equation is to be solved at various wave numbers or frequencies. Algorithms 1, 2 and 3 are high level description of the implementation of the pfft++.

```
Algorithm 1: construct Green's function Independent sparse matrices.
  Input: discretization, differential operator pairs
    ($\mathcal{F}_1$, $\mathcal{F}_2$) , interpolation stencil size , projection stencil size , direct stencil size
  Output: interpolation matrix $[I]$ and projection matrix $[P]$
  (1)  find the optimal grid size
  (2)  setup grid and element association
  (3)  setup interpolation stencil
  (4)  setup projection stencil
  (5)  setup direct stencil
  (6)  form the interpolation matrix $[I]$ for each $\mathcal{F}_1$
  (7)  form the projection matrix $[P]$ for each $\mathcal{F}_2$
```

```
Algorithm 2: construct Green's function dependent sparse matrices.
  Input: discretization, Green's function, differential
    operator pairs ($\mathcal{F}_1$, $\mathcal{F}_2$)
  Output: direct matrix $[D]$ and convolution matrix $H$
  (1)  form the sparse representation of $[H]$
  (2)  compute the FFT of $[H]$
  (3)  form the direct matrix $[D]$ for each pair of
       ($\mathcal{F}_1$, $\mathcal{F}_2$)
```

Using pfft++ to solve a single kernel integral equation such as (1) is straight forward. We could simply treat pfft++ as a black box that could perform the matrix vector product efficiently. After forming the four matrices by calling algorithms 1 and 2, algorithm 3 is to be called repeatedly in the inner loop of an iterative solver. To solve the integral equations with multiple kernels, we could simply repeat the above procedure for each integral operator individually.

4.7 Comparison to the original pFFT algorithm

The basic sparsification ideas in this paper are very similar to those in the original pre-corrected FFT algorithm [16]. The difference lies primarily in the ways the interpolation matrix and the projection matrix are generated. And this difference turns out to be important.

Algorithm 3: compute matrix vector product.
Input: vector x, differential operator pair ($\mathcal{F}_1$, $\mathcal{F}_2$)
Output: vector y

```
(1)    find the index $n$ of $[I]$ from $\mathcal{F}_1$
(2)    find the index $m$ of $[P]$ from $\mathcal{F}_2$
(3)    find the index $k$ of $[D]$ from operator pair
       ($\mathcal{F}_1$, $\mathcal{F}_2$)
(4)    $y_1 = [P_m]x$
(5)    $y_1 = fft(y_1)$
(6)    $y_2 = [H]y_1$
(7)    $y_2 = ifft(y_2)$
(8)    $y_3 = [I_n]y_2$
(9)    $y = y_3 + [D_k]x$
```

In the original pFFT algorithm [16, 25], the local collocation scheme is used to construct the projection matrix and the interpolation matrix is considered as the dual of the projection matrix. Hence both matrices are related to the Green's function or kernel. If one wants to solve a Helmholtz equation with different wave numbers or at different frequencies, these two matrices have to be re-generated for each frequency. As explained in section 4.6, the interpolation matrix and the projection matrix are only generated once in pfft++.

In the original pFFT algorithm, the convolution matrix is directly related to the kernel, which includes the effect of the operator $\mathcal{F}_2$. The convolution matrix in this work is directly related to the Green's function, not the operator $\mathcal{F}_2$. To see why this difference is important, suppose we want to compute the double-layer integral

$$\int_S d\vec{r}' \frac{\partial G(\vec{r},\vec{r}')}{\partial n(\vec{r}')} \rho(\vec{r}').$$

Using the original pFFT algorithm, it has to be done as the following

$$\int_S d\vec{r}' [n_x \frac{\partial G(\vec{r},\vec{r}')}{\partial x(\vec{r}')} + n_y \frac{\partial G(\vec{r},\vec{r}')}{\partial y(\vec{r}')} + n_z \frac{\partial G(\vec{r},\vec{r}')}{\partial z(\vec{r}')}] \rho(\vec{r}'). \tag{43}$$

This suggests that three convolution matrices $[H_x]$, $[H_y]$ and $[H_z]$ corresponding to $\frac{\partial G}{\partial x}$, $\frac{\partial G}{\partial y}$ and $\frac{\partial G}{\partial z}$ have to be generated and foreward FFT has to be performed for each of them. For each operation of the double-layer integral operator, $[H_x]\bar{\rho}$, $[H_y]\bar{\rho}$ and $[H_z]\bar{\rho}$ have to be carried out separately. As shown in section 4.3, pfft++ only needs one convolution matrix and hence only one convolution will be carried out in the matrix vector product step. This is a significant reduction in memory usage and CPU time.

5. Numerical Results

Base upon the algorithm described in section 4, we have developed pfft++, a flexible and extensible fast integral equation solver. The program pfft++ has been tested using random distributions on the surface of a sphere. After discretizing the surface, the integral operator in equation (1) is turned into either the dense matrix $[A]$ in (9) or the sparse matrix representation in (41). We assume a random vector α and compute the matrix vector product in (9) directly as $y_1 = [A]\bar{\alpha}$. We then compute the matrix vector product using pfft++ as $y_2 = pfft(\bar{\alpha})$. The relative error in the pFFT approximation is

$$error = (\frac{\sum_{i=1}^{N}(y_{1,i} - y_{2,i})^2}{\sum_{i=1}^{N} y_{1,i}^2})^{1/2}. \tag{44}$$

We first use a medium size example to demonstrate the trade-off between accuracy and CPU time and memory usage. We carried out the numerical experiment described above on a sphere discretized with 4800 panels.

Table 2: Relative error in (44) for different projection and interpolation stencil sizes and different kernels

	$p = 3$	$p = 5$	$p = 7$
$\frac{1}{r}$	$8.4e-5$	$1.3e-6$	$4.3e-9$
$\frac{\partial}{\partial n}\frac{1}{r}$	$8.5e-3$	$1.1e-4$	$8.4e-7$
$\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	$8.3e-5$	$1.3e-6$	$1.7e-9$
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	$6.0e-3$	$7.5e-5$	$5.9e-7$
$\frac{e^{ikr}}{r}$, $kR = 11.1$	$4.9e-4$	$1.1e-5$	$4.0e-7$
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 11.1$	$1.4e-2$	$2.8e-4$	$6.5e-6$

Table 3: CPU time for forming I, P, D and H matrices in (41) for different projection and interpolation stencil sizes and different kernels, unit is second

	$p = 3$	$p = 5$	$p = 7$
$\frac{1}{r}$	3.76	39.48	305.61
$\frac{\partial}{\partial n}\frac{1}{r}$	4.28	45.93	326.47
$\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	55.66	249.01	1022.05
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	47.80	229.02	971.32
$\frac{e^{ikr}}{r}$, $kR = 11.1$	53.06	242.65	1082.36
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 11.1$	47.99	226.89	967.58

When the kernels are Laplace kernel and its normal derivative, the radius of the sphere is $R = 1m$. When the kernels are Helmholtz kernel and its normal derivative, the radius of the sphere is $R = 5.3cm$ so that the size of the panels is smaller than one tenth of a wave length at 10GHz. Using increasingly larger stencil size in projection and interpolation, the accuracy is expected to increase. Table 2 clearly shows that this expectation has been met, where p stands for the stencil size of both projection and interpolation. For instance, $p = 3$ means that a $3 \times 3 \times 3$ 3D grid is used as the projection and the interpolation stencil. With the increase of the stencil size, the computational resource is expected to increase as well. This is shown in table 3, 4 and 5. The CPU time and memory usage increase significantly with the increase of the stencil size. In particular, the setup time of pfft++ increases by 4 to 10 times when stencil size increases from 3 to 5 or from 5 to 7. Though we only show data for a medium size problem here, from our numerical experiments, the observation is also true for large examples. Foutunately, almost all engineering problems only require modest accuracy, 3 to 4 digits. At this level of accuracy, the computational cost of pfft++ is very reasonable.

Figure 6 shows the CPU time versus problem size for different kernels. The projection and the interpolation stencil size is 3 for all these cases. It is clear that the CPU time grows almost linearly with the problem size for all types of kernels. Though not shown here in plot, the memory usage of pfft++ also grows linearly with the problem size.

6. Conclusions

This paper extends a recently developed pFFT algorithm to more general integral equations. Due to the introduction of polynomials in both interpolation and projection steps, pFFT now could handle complex kernels under a unified framework. It could also easily handle high-order elements as well as the discretization using Galerkin's method. A public-domain C++ code called pfft++ has been developed. Numerical results of large examples show that the memory usage and CPU time of the pfft++ are nearly $O(N)$.

Table 4: CPU time for doing one matrix vector product for different projection and interpolation stencil sizes and different kernels, unit is second

	$p=3$	$p=5$	$p=7$
$\frac{1}{r}$	0.07	0.11	0.17
$\frac{\partial}{\partial n}\frac{1}{r}$	0.07	0.11	0.17
$\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	0.20	0.33	0.64
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	0.20	0.33	0.61
$\frac{e^{ikr}}{r}$, $kR = 11.1$	0.19	0.32	0.63
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 11.1$	0.19	0.33	0.65

Table 5: Memory usage for different projection and interpolation stencil sizes and different kernels, unit is Mb

	$p=3$	$p=5$	$p=7$
$\frac{1}{r}$	10.75	35.18	87.94
$\frac{\partial}{\partial n}\frac{1}{r}$	10.75	35.18	87.94
$\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	16.04	47.3	114.5
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 1.11e-9$	16.04	47.3	114.5
$\frac{e^{ikr}}{r}$, $kR = 11.1$	16.04	47.3	114.5
$\frac{\partial}{\partial n}\frac{e^{ikr}}{r}$, $kR = 11.1$	16.04	47.3	114.5

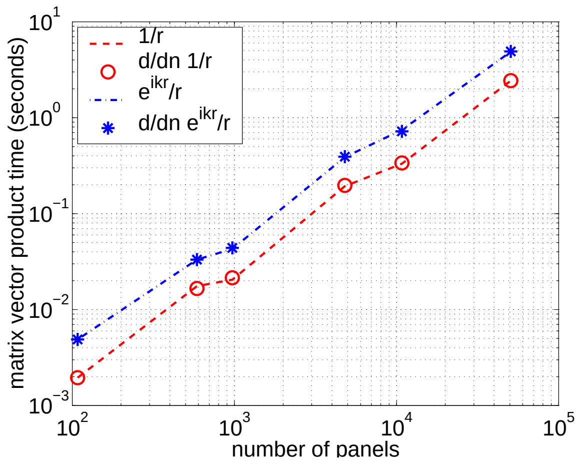

Figure 6: Run time of pfft++ versus the problem size, $kR = 11.1$ for Helmholtz kernel and its normal derivative

7. References

[1] K. Nabors and J. White, “FASTCAP: A multipole-accelerated 3-D capacitance extraction program,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, pp. 1447–1459, November 1991.

[2] M. Kamon, M. J. Tsuk, and J.K. White, “FastHenry: A multipole-accelerated 3-D inductance extraction program,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 42, no. 9, pp. 1750–1758, September 1994.

[3] J.M. Song, C.C. Liu, W.C. Chew, and S.W. Lee, “Fast illinois solver code (FISC) solves problems of unprecedented size at center for computational electromagnetics, university of illinois,” *IEEE Antennas and Propagation Magzine*, vol. 40, pp. 27–34, June 1998.

[4] C.C. Lu and W. C. Chew, “Fast algorithms for solving hybrid integral equations,” *IEEE Proceedings-H*, vol. 140, no. 6, pp. 455–460, December 1993.

[5] A.H. Juffer, E.F.F. Botta, B.A.M.Van Keulen, A.Van Der Ploeg, and J.C. Berendsen, “The electric potential of a macromolecule in a solvent: A foundamental approach,” *Journal of Computational Physics*, vol. 97, pp. 144–171, 1991.

[6] R.J. Zauha and R.S. Morgan, “The rigorous computation of the molecular electric potential,” *Journal of Computational Chemistry*, vol. 9, no. 2, pp. 171–187, 1988.

[7] J. Katz and A. Plotkin, *Low speed aerodynamics*, McGrath-Hill, New York, 1991.

[8] J.L. Hess and A.M.O. Smith, “Calculation of non-lifting potential flow about arbitrary three-dimensional bodies,” *Journal of ship research*, , no. 8, pp. 22–44, 1964.

[9] X. Wang, P. Mucha, and J.K. White, “Fast fluid analysis for multibody micromachined devices,” *Proceedings of MSM*, pp. 19–22, Hilton Head island, SC, 2001.

[10] L. Greengard and V. Rohklin, “A fast algorithm for particle simulations,” *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, December 1987.

[11] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*, M.I.T. Press, Cambridge, Massachusetts, 1988.

[12] V. Rokhlin, “Rapid solution of integral equations of scattering theory in two dimensions,” *J. Comp. Phys.*, vol. 86, pp. 414–439, 1990.

[13] V. Rokhlin, “Diagonal forms of translation operators for the Helmholtz equation in three dimensions,” *Applied and Computational Harmonic Analysis*, vol. 1, pp. 82–93, 1993.

[14] S. Kapur and J. Zhao, “A fast method of moments solver for efficient parameter extraction of MCMs,” 34th *ACM/IEEE Design Automation Conference*, pp. 141–146, 1997.

[15] Wolfgang Hackbush, *Integral Equations, Theory and Numerial Treatment*, Birkhauser Verlag, Basel, Switzerland, 1989.

[16] J. R. Phillips and J. K. White, “A precorrected-FFT method for electrostatic analysis of complicated 3D structures,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1059–1072, 1997.

[17] Kendall E. Atkinson, *The Numerial Solution of Integral Equations of the Second Kind*, Cambridge University Press, United Kingdom, 1997.

[18] O. Kellog, *Foundations of Potential Theory*, Frederick Ungar Publishing Corp., New York, 1929.

[19] R. Kress, *Linear Integral Equations*, Spring-verlag, world publishing Corp., Berlin; New York, 1989.

[20] Ronald B. Guenther and John W. Lee, *Partial differential Equations of mathematical Physics and Integral Equations*, Dover Publications, New York, 1996.

[21] Y. Saad, *Iterative methods for sparse linear systems*, PWS Publishing, Boston, MA, 1996.

[22] L.N. Trefethen and D. Bau, *Numerical linear algebra*, SIAM, Philadelphia, 1997.

[23] Youcef Saad and Martin Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM J. Sci. Statist. Comput.*, vol. 7, no. 3, pp. 856–869, July 1986.

[24] A. Brandt and A. A. Lubrecht, "Multilevel matrix multiplication and fast solution of integral equations," *Journal of Computational Physics*, vol. 90, pp. 348–370, 1990.

[25] Joel R. Phillips, *Rapid solution of potential integral equations in complicated 3-dimensional geometries*, Ph.D. thesis MIT EECS Department, 1997.

[26] G. H. Golub and C. F. Van Loan, *Matrix Computation*, The Johns Hopkins University Press, second edition, 1989.

[27] Bjarne Stroustrup, *The C++ Programming Language Special Edition*.

[28] Andrew Koenig and Barbara E. Moo, *Accelerated C++: Practical Programming by Example*.

[29] N. M. Josuttis, *The C++ Standard Library : A Tutorial and Reference*.