Parallel Fast Gauss Transform

Rahul S. Sampath
Oak Ridge National Laboratory
Oak Ridge, TN 37831
Email: sampathrs@ornl.gov

Hari Sundar
Siemens Corporate Research
Princeton, NJ 08540
Email: hari.sundar@siemens.com

Shravan K. Veerapaneni
New York University
New York, NY 10012
Email: shravan@cims.nyu.edu

***Abstract*—We present fast adaptive parallel algorithms to compute the sum of N Gaussians at N points. Direct sequential computation of this sum would take $\mathcal{O}(N^2)$ time. The parallel time complexity estimates for our algorithms are $\mathcal{O}\left(\frac{N}{n_p}\right)$ for uniform point distributions and $\mathcal{O}\left(\frac{N}{n_p}\log\frac{N}{n_p} + n_p \log n_p\right)$ for nonuniform distributions using n_p CPUs. We incorporate a plane-wave representation of the Gaussian kernel which permits "diagonal translation". We use parallel octrees and a new scheme for translating the plane-waves to efficiently handle nonuniform distributions. Computing the transform to six-digit accuracy at 120 billion points took approximately 140 seconds using 4096 cores on the Jaguar supercomputer at the Oak Ridge National Laboratory.**

Our implementation is *kernel-independent* and can handle other "Gaussian-type" kernels even when an explicit analytic expression for the kernel is not known. These algorithms form a new class of core computational machinery for solving parabolic PDEs on massively parallel architectures.

I. Introduction

Gauss transform is one of several discrete spatial transforms of the form

$$F(x_j) = \sum_{k=1}^{N} G_\delta(\|x_j - y_k\|) f(y_k) \quad \text{at} \quad \{x_j \,|\, j = 1, ..., M\}. \tag{1}$$

We refer to the points $x_j, y_k \in \mathbb{R}^3$ as the targets and sources respectively. The kernel G_δ is a smooth exponentially decaying function in both the physical and Fourier domains. We shall call such kernels as *Gaussian-type*. The parameter δ controls how rapidly the kernel decays. In the Gauss transform case, $G_\delta(\|x_j - y_k\|) = e^{-\frac{\|x_j - y_k\|^2}{\delta}}$.

Discrete sums of the form (1) are encountered in a variety of disciplines including computational physics, machine learning, computational finance and computer graphics [1], [2], [3], [4], [5]. The motivation for the present work comes from *potential theory* [6] applied to solving linear constant-coefficient parabolic partial differential equations (PDEs). Several advantages characterize potential theoretic approaches including reduction in dimensionality and exact satisfaction of the far-field conditions. A computational bottle-neck in such approaches is the need to evaluate discrete sums of the form (1). For example, high-order methods for the "heat potentials" require evaluating the convolutions [7], [8]

$$F(x) = \int_\Omega \|x - y\|^{2n} e^{-\frac{\|x-y\|^2}{\delta}} f(y)\, d\Omega \tag{2}$$

where Ω is the domain and n is a positive integer. Nyström method based on Gaussian quadrature for (2) gives rise to discrete sums of the form (1) because the kernel in (2) decays in both physical and Fourier domain [9]. More examples of Gaussian-type kernels can be found in [10]. If the kernel decays rapidly, one can simply truncate the sum to a few neighboring sources at each target. However, in many applications including heat potentials, this is not the case and computing these sums directly takes $\mathcal{O}(NM)$ time.

Related work. Because of their fundamental importance, fast schemes for (1) received significant attention in the recent past. Starting from the earlier work of Greengard and Strain [11], several sequential algorithms [12], [13], [14], [15], [9] have been proposed to reduce the cost to an optimal $\mathcal{O}(N+M)$. Recently, some effort in parallelizing has been made in [16] and [17]; the former in the context of radial basis function (RBF) interpolation using Gaussians and the latter for pricing weather derivatives. The scheme of [16] is based on truncating the sum locally and does not generalize to the case where the Gaussian spread is large. In [17], only smaller ($N, M = \mathcal{O}(100)$) one-dimensional problems were considered and they do not report scalability results beyond $n_p = 16$. To our knowledge, there have been no parallel implementations to date that compute (1) for highly nonuniform distributions and that are scalable in all parameter ranges.

Contributions. The main contributions of this work are summarized below.

- We describe a novel scheme for the translation of plane wave expansions; this is one of the steps in the sequential fast Gausss transform (FGT). Our new scheme reduces the storage and computational costs required for this step compared to previous implementations, especially for highly nonuniform point distributions.
- We present a parallel version of FGT for uniform distributions incorporating the accelerations introduced in [9] for tensor-product grids. We demonstrate the scalability of our algorithm using upto 120 billion points. To our knowledge, the Gauss transform of such a large number of points has not been computed before.
- We extend our algorithms to work with nonuniform distributions by using the linear octree data structure. This was motivated by the tree-splitting scheme used in [5] for computing continuous Gauss transforms. The cost of the original FGT algorithm increases as δ decreases; in contrast, our octree based algorithm scales well in all ranges of the parameters.
- We also present a parallel implementation of our octree based FGT algorithm.

 SC10 November 2010, New Orleans, Louisiana, USA 978-1-4244-7558-2/10/$26.00

The rest of the paper is organized as follows. In Section II, we present a short description of the standard FGT algorithm and discuss our parallel implementation of the same. We introduce a novel translation scheme in Section III. This scheme significantly improves the computational and storage costs for nonuniform distributions over the *sweeping algorithm* described in [12]. In Section IV, we present an octree-based FGT algorithm for nonuniform point distributions; we also discuss its parallel implementation. Finally, in Section V, we demonstrate the scalability of our algorithms for uniform as well as nonuniform point distributions. In the rest of the paper, we assume the following: (i) the sources and targets coincide, (ii) all points lie within the unit cube $[0,1]^3$, and (iii) the kernel G_δ is a Gaussian. These assumptions are purely for ease of exposition, all the algorithms are valid in the general case too.

TABLE I
FREQUENTLY USED PARAMETERS

FGT parameters	
δ	bandwidth of the kernel
h	size of an FGT box
ϵ	user specified precision
p	truncation order of the kernel expansion
K	length of interaction list in each dimension
$\|B\|$	number of non-empty FGT boxes
c^B	center of a FGT box B
Octree parameters	
ℓ	a leaf node of the octree
$\mathcal{Z}(s)$	Morton id of point/leaf s
m	upper bound for the number of points within any leaf
c	heuristic parameter used to classify leaves as either Expand or Direct
T_d	set of Direct leaves
T_e	set of Expand leaves
$\|N_d\|$	number of sources/targets in T_d
$\|N_e\|$	number of sources/targets in T_e

II. Overview of FGT for uniform distributions

The design of fast algorithms for (1) is strongly dependent on three independent parameters viz., the number of sources N, the bandwidth δ and desired accuracy ϵ. A Gaussian centered at a source location interacts with targets that are within its support. We use the *truncation algorithm* if the number of targets is less than a threshold value n^* and we use the *expansion algorithm*, otherwise. The threshold value depends on all three independent parameters and we will discuss its choice after introducing both algorithms.

A. Truncation algorithm

We introduce a few definitions first.

Definition 2.1 (Support of the kernel): The region around a target x where the kernel centered at x is greater than ϵ is called the support of the kernel and we denote it by ω.

Support of the Gaussian kernel, for instance, is the cube centered at x with side length $2\sqrt{\delta \ln(1/\epsilon)}$. We denote the volume enclosed by the ω as $|\omega|$.

Definition 2.2 (Interaction list): For any target point x, all the sources that are within the support of the kernel centered at x belong to the interaction list of x denoted by $\mathcal{I}[x]$.

In the truncation algorithm, we simply truncate the sum (1) to

$$F(x_j) = \sum_{y_k \in \mathcal{I}[x_j]} G_\delta(\|x_j - y_k\|) f(y_k). \tag{3}$$

There are $\mathcal{O}(N|\omega|)$ sources inside $\mathcal{I}[x_j]$ and hence, the complexity of this algorithm is $\mathcal{O}(N^2|\omega|)$. When the bandwidth δ is small and/or the ϵ is large, the support ω shrinks and the complexity approaches asymptotically to $\mathcal{O}(N)$. This is a common technique used in the graphics community (Gaussian blurring). When ω is comparable to the size of the domain (unit cube) or when fraction of points within the support grows beyond a threshold n^*, clearly, the cost of this algorithm grows quadratically. The expansion algorithm leads to optimal complexity in those cases. We discuss this next.

B. Expansion algorithm

Broadly speaking, FGT algorithms [11], [12] partition the domain into boxes of uniform size and proceed in three main steps: (i) compress the influence of sources into a few multipole-type moments, (ii) translate the moments across the domain and (iii) evaluate the moments at the targets. In the original FGT [11], the translation costs tend to dominate [9]. The plane-wave expansion version [12] minimized the translation costs to a large extent but at the expense of higher computational cost for steps (i) and (iii). This problem is alleviated in [9] to a large extent. Our implementation is based on [9] and we summarize it here.

Central to the expansion algorithm is the finite-term plane-wave expansion of the kernel,

$$G_\delta(\|x_j - y_k\|) \approx \sum_{|k| \le p} \hat{G}(k) e^{i\lambda k \cdot (x_j - y_k)}, \quad \lambda = \frac{L}{p\sqrt{\delta}} \tag{4}$$

where $k = (k_1, k_2, k_3)$ and the parameters p and L are determined by the required precision. $\hat{G}$ is the discrete Fourier transform of the kernel. For the Gaussian kernel, it is given by

$$\hat{G}(k) = \left(\frac{L}{2p\sqrt{\pi}}\right)^3 e^{-\frac{\lambda^2 |k|^2 \delta}{4}}. \tag{5}$$

The algorithm begins by partitioning the domain into uniform boxes of size $\sqrt{\delta}$ each. The total number of boxes denoted by $|B|$ are then given by $|B| = (1/\sqrt{\delta})^3$. The kernel located at the center of a box B decays below ϵ beyond a fixed number of boxes. We denote this fixed number in 1D by K (K^d in d dimensions).

Definition 2.3 (Interaction list): The interaction list of a box B denoted by $\mathcal{I}[B]$ is the set of all boxes that are covered by the support of the kernel located at the center of B.

Using these definitions, we are now ready to state the algorithm. A target $x \in D$ receives information from a source $y \in B$ in three steps:

S2W The influence of all the sources in the box B is condensed into a plane-wave expansion:

$$w_k = \sum_{y \in B} f(y) e^{i\lambda k \cdot (c^B - y)} \quad \forall \quad |k| \le p. \tag{6}$$

W2L The plane-wave expansion of each box B is transmitted to every box D in $\mathcal{I}[B]$. By superposition, the "local" plane-wave expansion of D, denoted by v_k for $|k| \leq p$, gets modified as

$$v_k += w_k e^{i\lambda k \cdot (c^D - c^B)}. \tag{7}$$

L2T The transform (1) is computed at each target by evaluating the local expansion of the box it is contained in:

$$F(x) = \sum_{|k| \leq p} \hat{G}(k) v_k e^{i\lambda k \cdot (x - c^D)}. \tag{8}$$

First, the S2W step is executed separately in each box with $\mathcal{O}(p^3 N)$ work. In a *direct translation scheme*, the W2L is executed by simply visiting each box B and *translating* its plane-wave expansion to all the boxes in $\mathcal{I}[B]$. Assuming the size of interaction list is K^3, this algorithm requires $\mathcal{O}(K^3 p^3 |B|)$ work to form local expansions at all the boxes. Finally, the L2T is executed by visiting each box and evaluating the local expansion at the targets, requiring $\mathcal{O}(p^3 N)$ work. Therefore, the overall cost of the algorithm is $\mathcal{O}(p^3 N + K^3 p^3 |B|)$.

C. Overall algorithm

The translation costs can be reduced dramatically by using the *sweeping algorithm* introduced in [12]. We present a modified version of the same in Section III. The total cost of the expansion algorithm is then typically dominated by the S2W and L2T steps which grow as $\mathcal{O}(p^3 N)$. However, when δ is lower and/or ϵ is higher, the cost of the expansion algorithm increases because condensing the sources into plane-waves becomes increasingly futile as the number sources per box gets reduced. On the other hand, the cost of the truncation algorithm decreases. Hence, we choose between the two algorithms based on the following heuristic:

```
if N|ω| < (2p)^3 then
  Use truncation algorithm
else
  Use expansion algorithm
end if
```

The complexity of the overall algorithm, therefore, is $\mathcal{O}(p^3 N)$ independent of δ (or $|\omega|$).

Kernel independence. All the steps we have described so far are applicable for any Gaussian-type kernel. In fact, except the L2T step which requires the Fourier transform of the kernel, the formulas in the remaining steps are the same for any kernel. Given the kernel values at the discrete points $\{Lk/p\}_{|k| \leq p}$, we can compute $\hat{G}$ using the discrete Fourier transform. The parameters L, p and K, however, need to be tuned for desired accuracy based on the kernel [9].

D. Parallel truncation algorithm

We divide the unit cube into a regular grid of cuboidal blocks and distribute the blocks across n_p processors such that *a*) each processor owns exactly one block, and, *b*) each block is owned by an unique processor. For each target owned by a particular processor, we find the processors that contain blocks which intersect the interaction list of that target and send the source information to those processors. Each processor then evaluates (3) at all its targets.

E. Parallel expansion algorithm

We create a regular grid of $|B|$ FGT boxes distributed on n_p processors such that `(a)` each box is owned by an unique processor and `(b)` each processor owns a sub-grid of FGT boxes. For ease of implementation, we assume that $|B| > n_p$. We use PETSc's [18] DA module to manage this distributed regular grid. The S2W and L2T steps are embarrassingly parallel; in the former step, each processor independently forms the plane-wave expansions for each box that it owns and in the latter step, each processor independently computes the transform (1) for each target contained within some box owned by that processor using the local expansion for that box. Hence, the cost for these two steps is simply $\mathcal{O}(p^3 \frac{N}{n_p})$. For the W2L step, each processor first gathers the plane-wave expansions of the boxes that are in the interaction list of some box owned by that processor. The communication cost for this step is $\mathcal{O}(K(\frac{|B|}{n_p})^{\frac{2}{3}} + K^2(\frac{|B|}{n_p})^{\frac{1}{3}} + K^3)$. Following this communication, each processor executes the sequential W2L algorithm for all boxes that it owns. If we use the direct scheme for the sequential W2L algorithm then this cost would be $\mathcal{O}(p^3 K^3 \frac{|B|}{n_p})$. As mentioned earlier, this cost can be reduced by using the sweeping algorithm, which is described in the following section.

III. Translation scheme

Translation, as we have seen in Section II, is forming the local expansion at a target box by combining the plane-wave expansions of all source boxes in its interaction list. The cost of the direct scheme (7) is $\mathcal{O}(K^3 p^3 |B|)$. For uniform distributions, the *sweeping algorithm* introduced in [12] and extended to multi-dimensions in [9] brings down the complexity to $\mathcal{O}(9p^3 |B|)$.

For nonuniform distributions, we do not form plane-wave expansions at all the boxes (we give more details in the Section IV). For example, if there are no sources in a particular box, there is no need for executing the S2W step in this box. We shall call the boxes where the plane-wave expansions are formed as *non-empty* and the others as *empty*. Predominantly, there are two kinds of non-empty box distributions:

- Volume distribution, in which, each source box on an average has $\mathcal{O}(K^3)$ target boxes in its interaction list. They arise from highly nonuniform volumetric distribution of sources resulting in empty boxes wherever there are no sources within them. Note that the uniform distribution also falls into this category.
- Surface distribution, in which, each source box on an average has $\mathcal{O}(K^2)$ target boxes in its interaction list. These arise when the sources reside on two-dimensional manifolds (eg., a sphere) embedded in the volume.

Suppose the unit cube $[0,1]^3$ is partitioned into n_b^3 regular boxes and $|B|$ of them are non-empty. There are two main drawbacks to the sweeping algorithm. First, a naïve implementation stores data in all the n_b^3 boxes. This could be detrimental

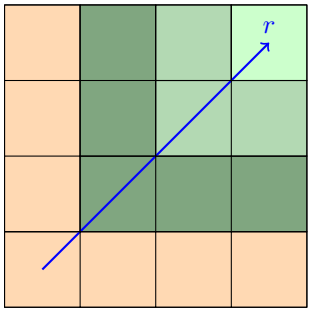


Fig. 1. The outermost layer is shown in orange, for which the local expansions are computed directly. The other layers (in green) are computed by propagation. The direction of propagation (r) is shown using the arrow and by the shade of the layers, lighter being later in time.

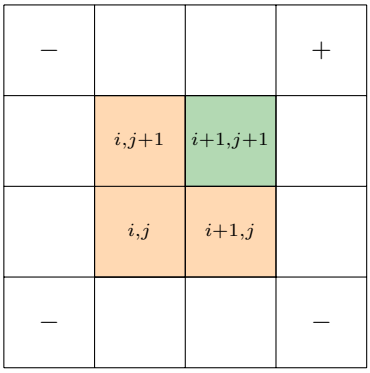


Fig. 2. Illustration of the modified sweeping. The local expansions of the boxes shown in orange color are used for computing the local expansion at the box shown in green color. The stencil is given in equation (9). Assuming $K = 3$, a simple inspection of the interaction lists reveal that the contributions of boxes marked by "+" must be added and those marked by "-" must be substracted.

both for the volume and the surface distributions as, potentially, $|B| << n_b^3$. Second, its performance is worse than the direct scheme for surface distributions. For example, consider the case where the sources reside on the sphere. The sweeping algorithm fills in $\mathcal{O}(K^3)$ empty boxes around each non-empty box increasing the count of non-empty boxes from $|B|$ to $\mathcal{O}(K^3|B|)$. Thereby, the overall cost of sweeping algorithm is $\mathcal{O}(9p^3K^3|B|)$. The direct scheme, on the other hand, requires $\mathcal{O}(p^3K^2|B|)$ because each source box has only $\mathcal{O}(K^2)$ target boxes in its interaction list.

In this section, we propose two algorithms, one for volume and the other for surface distributions, that achieve optimal complexity and have minimal storage requirements. Given any arbitrary source distribution, it could be hard to distinguish which category the resulting non-empty box distribution falls into. We present a computationally inexpensive method to compute the cost of both algorithms *a priori* and pick the optimal of the two.

A. Modified sweeping algorithm for volume distributions

Instead of sweeping along each dimension one-by-one as proposed in [9], our algorithm visits each box only once and computes the full plane-wave translation based on its neighbors which have already been visited. The plane-wave expansion at any particular box is represented as a combination of the plane-wave expansions of its $2^d - 1$ neighbors, along with corrections for 2^d boxes. We start with direct computation of the outermost layer, and then propagate it inwards, layer by layer, as illustrated in Figure 1. This results in an overall complexity of $\mathcal{O}((2^{d+1}-1)p^d|B|)$. The constant in the complexity for this step increases from $3d \rightarrow 2^{d+1} - 1$, which is from $6 \rightarrow 7$ for $d = 2$ and from $9 \rightarrow 15$ for $d = 3$. On the other hand, this enables us to reduce the storage cost from $\mathcal{O}(n_b^3)$ to $\mathcal{O}(|B| + n_b^2)$.

The algorithm is illustrated in Figure 2. The steps involved in the algorithm are as follows,

1) First compute the local expansions at the outermost layer of the FGT boxes. Full computation with a complexity $\mathcal{O}(K^d p^d)$ is only required for the first box, as the local expansions for the other boxes can be computed by adding and subtracting layers from the expansions of the already computed boxes.
2) We propagate the local expansions from this initial layer to subsequent layers. Consider the case shown in Figure 2 where we need to compute the local expansion of the Green box ($B(i+1, j+1)$), given the local expansions of the adjacent boxes (colored in orange).
3) The local expansion $\{v_k^{(i+1,j+1)}, |k| \leq p\}$ of the box $B(i+1, j+1)$ can be written in terms of the local expansions of the $2^d - 1$ neighbors $B(i, j), B(i+1, j)$ and $B(i, j+1)$, along with corrections terms from the 2^d corners. These are marked in Figure 2 assuming $K = 3$. Consider the box $B(i+1, j+1)$ with 2^d-1 neighbors at offsets ξ and having 2^d corners at offsets χ. The local expansion can therefore be computed as,
$$v_k^{(i+1,j+1)} = \sum_{\xi} (-1)^{1+d+\mathcal{M}(\xi)} e^{i\lambda k\cdot\xi} v_k^{(i,j)+\xi} + \sum_{\chi} (-1)^{1+d+\mathcal{M}(\chi)} e^{i\lambda k\cdot\chi} w_k^{(i,j)+\chi}. \quad (9)$$
The offsets χ, ξ represent the offsets of neighbors or the corners with respect to $B(i+1, j+1)$. $\mathcal{M}(\chi), \mathcal{M}(\xi)$ is the ordinal of the contributing neighbor or corner, which in turn determines whether the contribution is positive or negative. $\mathcal{M}(\chi)$ is the number of dimensions in which χ is different from the top-right (most extreme in n-D) corner or neighbour. In the example shown in Figure 2, the ordinal of $B(i+1, j+1)$ is 0 and that of $B(i, j)$ is 2.
4) The propagation can then be used to compute the local expansions at the remaining boxes in the new propagation layer. At any given stage only the values of the current propagation layer needs to be stored. This reduces the storage cost for the new algorithm from $\mathcal{O}(n_b^3)$ to $\mathcal{O}(|B|+n_b^2)$.

B. Modified sweeping algorithm for surface distributions

For surface distributions, the algorithm introduced in the previous section suffers from the same problem as the sweeping algorithm in that it generates large number of additional non-empty boxes and consequently it becomes computationally more

expensive than the direct scheme. We introduce a new approach here that does not generate any extra non-empty boxes and performs better than the direct scheme.

The main idea, however, is similar: once a local expansion is formed at a particular box, we use this information to minimize the translation cost at a subsequently visited box. Let us assume that the boxes are ordered as $\{B_k | k = 1, \ldots, |B|\}$. At the first box B_1 we form the local expansion directly with $\mathcal{O}\left(|\mathcal{I}[B_1]|\, p^3\right)$ work, where $|\cdot|$ is the size of the set. For subsequent boxes, we only need to add/subtract the plane-wave expansions of the boxes that are not present in both the interaction lists of the current box and that of the previously visited box. The problem is therefore to determine the optimal traversal such that the overall cost is reduced.

Problem 3.1 (Optimal Traversal): Find an ordering $\{B_1, B_2, \ldots, B_{|B|}\}$ *so that*

$$|\mathcal{I}[B_1]| + \sum_{j=1}^{|B|-1} |\mathcal{I}(B_j) \cup \mathcal{I}(B_{j+1}) \setminus \mathcal{I}(B_j) \cap \mathcal{I}(B_{j+1})| \quad (10)$$

is minimized.

The first term in equation (10) is the translation cost of the first box and the term inside the summation encodes the cost of translating the local expansion from B_j to B_{j+1}. Note that cost of translating plane-waves between two consequent boxes in the ordering is minimal if their interaction lists have many common elements. There are many ways to find the optimal traversal path. We take a graph theoretic approach by reducing optimal traversal problem to one of finding the minimum spanning tree of a weighted graph.

Problem 3.2 (MST): Construct a weighted graph $G = (V, E)$ *with the non-empty FGT boxes as its vertices. An edge is introduced between vertices* v_i *and* v_j *if and only if* $B_i \in \mathcal{I}[B_j]$. *The weight of an edge between boxes* B_i *and* B_j *is given by*

$$w_{ij} = |\mathcal{I}(B_i) \cup \mathcal{I}(B_j) \setminus \mathcal{I}(B_j) \cap \mathcal{I}(B_j)|. \quad (11)$$

Introduce an additional vertex $v*$ *which connects to every vertex* v_i *in the graph by an edge whose weight is equal to* $|\mathcal{I}[B_i]|$. *Find the minimum spanning tree of* G.

The vertex $v*$ encodes the cost of the direct computation at the boxes. We compute the minimum spanning tree of this graph using Kruskal's algorithm [19] whose complexity is $\mathcal{O}(|E| \log |V|)$, which in our case becomes $\mathcal{O}(K^2|B| \log |B|)$ because $|E| = K^2|B|$ and $|V| = |B|$. From a practical perspective, it is sufficient to add the edges for the immediate neighbors of a vertex v_i, resulting in $|E| = 13|B|$ in 3D.[1] This approach brings down the complexity of determining the optimal path to $\mathcal{O}(|B| \log |B|)$.

The traversal is then defined by the tree rooted at $v*$. Depending on how the data is stored, the tree can be traversed in a depth-first or a breadth-first fashion. In our implementation, we use a breadth-first traversal. Once the optimal ordering is computed by solving the Problem 3.2, the traslation step is carried out by traversing the tree and forming the local expansion at the box B_{j+1} using the local expansion at B_j and by substracting the contributions of boxes in

$$\mathcal{I}[B_j] \setminus \mathcal{I}(B_j) \cap \mathcal{I}(B_{j+1})$$

and adding the contributions of boxes in

$$\mathcal{I}[B_{j+1}] \setminus \mathcal{I}(B_j) \cap \mathcal{I}(B_{j+1}).$$

The average time complexity of this scheme is $\mathcal{O}(Kp^3|B|)$. We demonstrate the speed-ups achieved for non-empty box distributions on a sphere in Table II, Figures 4 and 3.

A priori cost estimation. For a given non-empty box distribution, we can estimate the cost of the propogation algorithm described in Section III-A by visiting each non-empty box and

[1]This can result in an increased cost due to direct computation for isolated boxes.

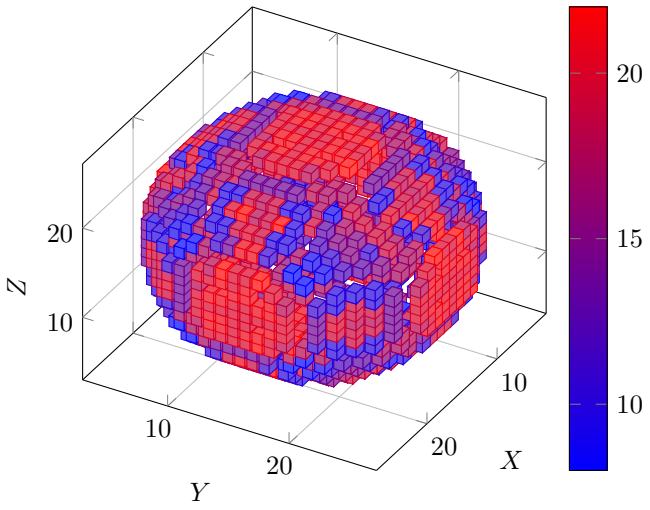


Fig. 3. The cost of translations for the optimal traversal path for a spherical shell distribution ($K = 9$). The boxes B_i are color coded based on the cost of performing the translation. For comparison the average cost for direct translations in this case is 99.

TABLE II
THE REDUCTION IN THE COMPLEXITY OF COMPUTING THE PLANE-WAVE TRANSLATIONS BY USING OPTIMAL TRAVERSAL COMPARED TO DIRECT COMPUTATIONS. RESULTS ARE PRESENTED FOR $K = 9$ AND $K = 13$. IT CAN BE SEEN THAT THE SPEEDUP IS PROPORTIONAL TO K AND SCALES WITH $|B|$.

$\|B\|$	direct	MST	ratio
$K = 9 \quad (\epsilon = 10^{-6})$			
1K	114K	18K	6.3
10K	1M	182K	5.88
100K	9.65M	1.7M	5.68
1M	96.8M	17.1M	5.68
$K = 13 \quad (\epsilon = 10^{-12})$			
1K	238K	24K	9.65
10K	2.24M	264K	8.48
100K	20.16M	2.46M	8.18
1M	200.56M	24.5M	8.18

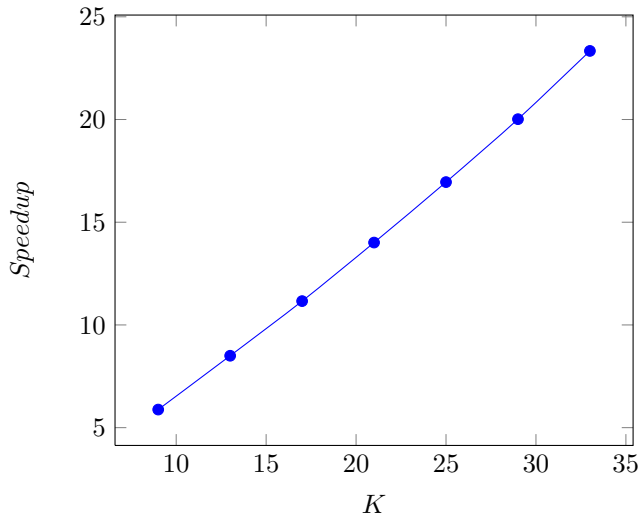


Fig. 4. Speedup obtained for increasing values of the size of the interaction list, K, for a fixed geometry. The speedup is proportional to K and the complexity of computing the translations is reduced from $\mathcal{O}(K^2|B|)$ to $\mathcal{O}(K|B|)$.

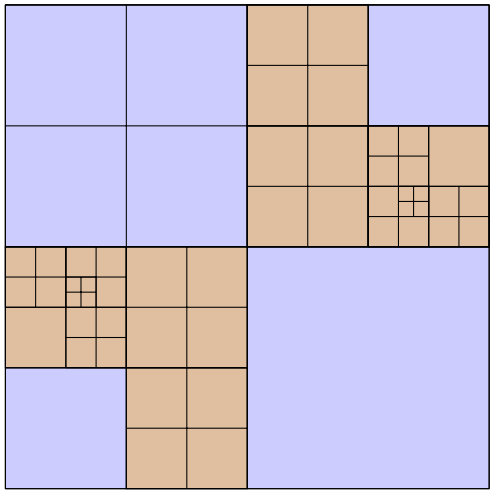


Fig. 5. An example of a quadtree (2D analogue of octree) whose leaf nodes are distinguished based on their size. The "large" leaf nodes shown in blue color belong to T_d and the "small" leaf nodes shown in brown color belong to T_e.

counting the number of empty boxes in its interaction list. The worst case complexity of this estimation algorithm is $\mathcal{O}(K^3|B|)$. The operation count for the W2L step via propogation algorithm is $15p^3|B_{\text{total}}|$ where $|B_{\text{total}}|$ is the number of non-empty boxes plus the number of empty boxes in their interaction lists.

The operation count for the W2L step via MST algorithm is $p^3\mathcal{C}$ where $\mathcal{C}$ is given by (10). This can be estimated by solving Problem 3.2. Therefore, with a worst case complexity of $\mathcal{O}\left((K^3 + \log|B|)|B|\right)$, we can precisely estimate the operation counts for both the algorithms and chose the one with the lowest cost.

IV. FGT for non-uniform distributions

In the case of uniform point distributions, we could obtain precise estimates for the threshold n^* and use it to choose between the truncation and expansion algorithms to achieve optimal complexity. However, when the source and target distributions are highly non-uniform, as is the case in most practical applications, it is not as straightforward. Note that the expansion algorithm is advantageous when there are many sources within a FGT box and the truncation algorithm is more suitable otherwise. Hence, to achieve optimal complexity in the non-uniform case, we need a hybrid algorithm that uses the expansion algorithm in regions with a high density of points and the truncation algorithm in other regions. We use an *octree* [20] data structure to efficiently separate out regions with a dense point distribution from regions with a sparse point distribution and use it to appropriately choose between the truncation and expansion algorithm in each region. A similar idea was used in [5].

Primer on octrees. An octree is a tree data structure that is used for spatial decomposition. Every node of an octree has a maximum of eight children. An octant with no children is called a *leaf* and an octant with one or more children is called an *interior octant*. The only octant with no parent is the *root* and all other octants have exactly one parent. The depth of an octant from the root is referred to as its *level*. We use a *linear* octree representation (i.e., we exclude interior octants) using the *Morton encoding* scheme [21]. Any octant in the domain can be uniquely identified by specifying one of its vertices, also known as its *anchor*, and its level in the tree. By convention, the anchor of an octant is its front lower left corner.

A. Octree based FGT algorithm

Given a point distribution, we construct a linear octree such that there are no more than m points contained within each leaf; we use the algorithm described in [22] to do this. Note that regions with sparse point distributions will contain leaves that are large in size and regions with dense point distributions will contain leaves that are small in size. Just as in the uniform case, we partition the domain into FGT boxes of size $h = \sqrt{\delta}$. We denote the size of leaf ℓ by $|\ell|$. We use a heuristic parameter, c, to mark leaves as either "Expand" or "Direct" as follows (Figure 5):

```
for each leaf ℓ do
  if |ℓ| > ch then
    Mark ℓ as ``Direct''.
  else
    Mark ℓ as ``Expand''.
  end if
end for
```

We denote the set of Direct leaves by T_d and use T_e to represent the set of Expand leaves. In order to achieve optimal complexity, the orginial FGT [11] chooses different schemes for transmitting the source information to the targets depending on the sources and target distributions. It, however, did not consider the case where δ is small, wherein, it will generate a large number of boxes and thereby incur higher computational cost. An advantange of our octree based algorithm is that all

these cases including the small δ case can be efficiently handled with just the one spatial decomposition into Direct and Expand leaves. We first summarize how the information is transmitted from a source y to a target x and then describe in detail all the steps involved in this algorithm.

y	x	
T_e	T_e	$y \xrightarrow{\textbf{S2W}} \{w_k\} \xrightarrow{\textbf{W2L}} \{v_k\} \xrightarrow{\textbf{L2T}} x$
T_e	T_d	$y \xrightarrow{\textbf{S2W}} \{w_k\} \xrightarrow{\textbf{W2D}} x$
T_d	T_e	$y \xrightarrow{\textbf{D2L}} \{v_k\} \xrightarrow{\textbf{L2T}} x$
T_d	T_d	$y \xrightarrow{\textbf{D2D}} x$

S2W In this step, we form the plane-wave expansions (6) at all the boxes by visiting every leaf node in T_e. A particular leaf ℓ can either be contained within a box B or can enclose several boxes.[2] In the former case, we add the contributions from all sources within ℓ to the plane-wave expansion at B. In the latter case, we visit each constituent box B and form its plane-wave expansion using only those sources that are contained in B.

W2D In this step, the influence of all the sources in T_e is accumulated at all the targets in T_d. First, we initialize the transform at all targets in T_d to zero. Then, for each target $x \in T_d$, we find all FGT boxes, B, such that $\mathcal{I}[B]$ contains x. We add the contribution of sources in B to the transform at x using its plane-wave expansion, w_k:

$$F(x) += \sum_{|k|\leq p} \hat{G}(k) w_k e^{i\lambda k\cdot(x-c^B)} \tag{12}$$

Note that for any FGT box B that is contained within some Direct leaf, the corresponding w_k is zero.

D2D For each source $y \in T_d$, find all targets $x \in T_d$ such that $x \in \mathcal{I}[y]$ and modify the transform at x as:

$$F(x) += G_\delta(\|x-y\|) f(y) \tag{13}$$

We make use of a few properties of the Morton ordering to search for the targets $x \in T_d \cap \mathcal{I}[y]$ more efficiently. These are listed below:

- Let $r = \sqrt{\delta \ln(\frac{1}{\epsilon})}$, $y_{min} = y - (r,r,r)$ and $y_{max} = y + (r,r,r)$. Then, the Morton id of any point in $\mathcal{I}[y]$ will be greater than or equal to the Morton id of y_{min} and less than or equal to that of y_{max}.
- If a point z is contained within a leaf ℓ, then the Morton id of z will be greater than that of ℓ.
- If the Morton id of a leaf l_1 is less than that of another leaf l_2, then the Morton id of any point, z, which is contained within l_1 will also be lesser than that of l_2.

We make use of the above properties along with the fact that T_d is sorted in the Morton ordering and search for the leaves that contain potential targets rather than searching for the targets directly. After shortlisting the set of candidate leaves, we then search for the relevant targets within them.

W2L Only the boxes that share sources with atleast one Expand leaf node are non-empty, that is, carry a plane-wave expansion. Based on the distribution of the non-empty boxes, we choose between the modified sweeping algorithms for surface or volume distributions as discussed in Section III.

D2L This step is the dual of the W2D step. Here, we accumulate the influence of sources in T_d at the targets in T_e. For each source $y \in T_d$, we find each FGT box, D, that lies within the support of the kernel centered at y and modify its local expansion as:

$$v_k += f(y) e^{i\lambda k\cdot(c^D - y)} \tag{14}$$

L2T For each target $x \in T_e$, we evaluate the transform using the local expansion of the FGT box, D, that contains x using (8).

B. Parallel Implementation

We create a regular grid of FGT boxes partitioned across processors such that `(a)` the size of each box is $h = \sqrt{\delta}$, `(b)` each processor owns a sub-grid of boxes and `(c)` each box is owned by an unique processor. We use `PETSc`'s [23], [18] DA module to manage this distributed regular grid. We construct a parallel linear octree such that each leaf contains fewer than m points; we use `DENDRO` [24] to do this. We then mark the leaves as either "Expand" or "Direct"; this step is embarrassingly parallel. The Direct leaves are then partitioned across the processors such that `(a)` they are globally sorted in the Morton ordering and `(b)` each leaf is owned by an unique processor. Similarly, we partition the Expand leaves across the processors. The Expand and Direct leaves are used at different steps in the algorithm and there are collective communications between these steps. So, it is better to have good load balance for the Expand and Direct leaves independently and hence we partition the two sets, T_d and T_e, independently.

The plane-wave expansions in the S2W step can be computed independently by each processor, without any communication. However, the Expand leaves and their corresponding FGT boxes may belong to different processors. So, the plane-wave expansions computed in the S2W step must be sent to the processors that own the corresponding FGT boxes. The owner of a FGT box will then add the values it receives from other processors to its existing plane-wave expansions. We refer to this step as **S2W-Comm**.

In the W2D step, each processor first forms a list of all FGT boxes whose interaction list contains at least one target, $y \in T_d$, owned by this processor. These boxes are then sent to the respective processors that own them. The owner of an FGT box

[2] Since the box size is chosen as $\sqrt{\delta}$, in the general case, there could be cases where boxes and leaf nodes overlap partially. Here, we make a simplifying assumption that $\delta = 2^{-n}$ for some positive integer n. This assumption can be relaxed easily by setting the box size to be $2^{\lfloor \log_2 \sqrt{\delta} \rfloor}$ and modifying the error estimates for FGT appropriately.

returns the plane-wave coefficients for that box to the processors that requested them. After this communication, each processor independently computes (12).

Next, gather the Morton id of the first (in the sorted list) Direct leaf on each processor. Let this list be denoted by S; this will be used to identify the processor that owns a given leaf/point. Each processor then sends each source, y, that lies within its portion of T_d to those processors that contain Direct leaves with Morton ids that overlap with the range $[\mathcal{Z}(y_{min}), \mathcal{Z}(y_{max})]$, where y_{min} and y_{max} are defined as in Section IV-A. We identify these processors by first finding s_1 and s_2 such that s_1 is the greatest value in S that is $\leq \mathcal{Z}(y_{min})$ and s_2 is the greatest value in S that is $\leq \mathcal{Z}(y_{max})$ and then selecting all processors that contain Direct leaves with Morton ids that lie in $[s_1, s_2]$. Each processor then computes (13) independently.

We then communicate the plane-wave expansions of the *ghost* boxes[3] just like in the W2L step of the parallel expansion algorithm (Section II-E). Subsequently, each processor independently executes the rest of the W2L step as described in Section IV-A.

In the D2L step, each processor first computes its contributions to the local expansions of those FGT boxes that intersect the interaction list of at least one source contained within some Direct leaf owned by this processor; this does not require any communication. Next, we send these contributions to the processors that own the respective FGT boxes. The owner of an FGT box will then add the values it receives from other processors to its existing local expansions, v_k.

The local expansions for each FGT box that overlaps at least one Expand leaf are then sent to the processors that own the corresponding Expand leaf. We refer to this step as **L2T-Comm** and it is like the dual of the S2W-Comm step.

Finally, each processor independently executes the L2T step and computes the transform at all targets within its portion of T_e.

We summarize the overall algorithm and give the complexity estimates for the main steps in Algorithm 1.

Algorithm 1 *Parallel FGT for non-uniform distributions*

Input: N Points, δ, ϵ, m and c
1. Create a regular grid of FGT boxes partitioned across processors such that
(A) the size of each box is $h = \sqrt{\delta}$,
(B) each processor owns a sub-grid of boxes and
(C) each box is owned by an unique processor.
2. Construct a linear octree such that each leaf contains fewer than m points. $\mathcal{O}(\frac{N}{n_p}\log\frac{N}{n_p} + n_p \log n_p)$
3. Mark each leaf as either ``Expand'' or ``Direct'' based on c and δ.
4. Partition the Direct leaves across processors such that
(A) they are globally sorted in the Morton ordering and
(B) each leaf is owned by an unique processor.
5. Partition the Expand leaves across processors.
6. Execute S2W. $\mathcal{O}(p^3\frac{N}{n_p})$
7. Execute S2W-Comm. $\mathcal{O}(n_p + p^3\frac{|B|}{n_p})$
8. Execute W2D.
9. Execute D2D.
10. Execute W2L. $\mathcal{O}(p^3\frac{|B|}{n_p} + K(\frac{|B|}{n_p})^{\frac{2}{3}} + K^2(\frac{|B|}{n_p})^{\frac{1}{3}} + K^3)$
11. Execute D2L.
12. Execute L2T-Comm. $\mathcal{O}(n_p + p^3\frac{|B|}{n_p})$
13. Execute L2T. $\mathcal{O}(p^3\frac{N}{n_p})$

V. Results

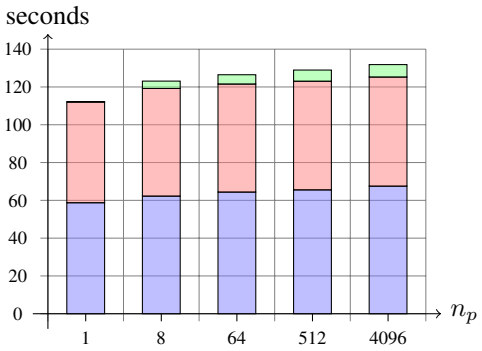
In this section, we present scalability results using two test cases. All the scalability experiments were performed on the *Jaguar* supercomputer at Oak Ridge National Laboratory (ORNL). The architectural details for this supercomputer can be found in [25]. The code is written in the C++ language and built on top of `PETSc` and `DENDRO` packages. Specifically, we use `DENDRO` for managing linear octrees and `PETSc` for managing distributed regular grids and profiling.

Example 1. In this example, we consider a uniform tensor-product grid obtained by using Gaussian quadrature rule to compute (2) within the unit cube. The source strength $f(y)$ is chosen randomly and the targets are same as the sources. We used the expansion algorithm and incorporated the acceleration techniques introduced in [9] for tensor-product grids. They are based on *separation of variables* and the complexity of the S2W and L2T steps is reduced from $\mathcal{O}(p^3N)$ to $\mathcal{O}(pN)$.

We report the weak scalability results in Figure 6. The grain size is fixed to approximately 30 million sources/targets per CPU. We also reduce δ so that the number of FGT boxes per CPU remains fixed. We get excellent scalabilty in this case–only a small increase in timing as we go from 1 to 4096 CPUs. This is because the cost is dominated by the S2W and L2T steps and both of them are embarassingly parallel in our implementation. There is only one communication step in the algorithm: a point-to-point communication in the W2L step for communicating the plane-wave expansions of ghost boxes.

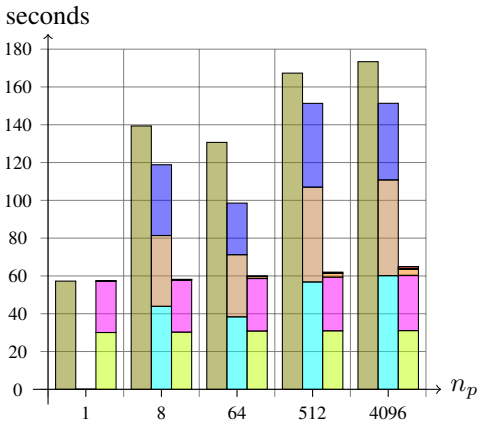
Example 2. In this example, we consider a Gaussian random distribution of points in the unit cube. We construct a linear octree from these points such that each leaf contains 7^3 sources/targets within it. We used our octree-based FGT algorithm. We present the weak scalability results in Figure 7. We varied the number of input points such that the number of expand leaves per CPU and the number of direct leaves per CPU remain approximately constant as we vary the number of CPUs. There is a significant increase in the timings between the 1 CPU and 8 CPU cases because there were no direct leaves

[3]We refer to any box that is owned by a different CPU but lies in the interaction list of some box owned by this CPU as a ghost box.



S2W	58.75	62.25	64.39	65.56	67.53
L2T	53.36	56.98	57.18	57.50	57.75
W2L	0.0698	3.90	4.93	5.92	6.59
Total	**112.19**	**120.93**	**125.06**	**127.62**	**131.65**
Points	29.2M	233.8M	1.9B	15.0B	119.7B
δ	$\frac{1}{16}$	$\frac{1}{64}$	$\frac{1}{256}$	$\frac{1}{1024}$	$\frac{1}{4096}$

Fig. 6. Isogranular scalability for an uniform point distribution. For this experiment, we set $\epsilon = 10^{-6}$. The reported times for each component are the maximum values for that component across all the processors. The total wall-clock time is reported in bold face.



D2L	0	43.93	38.37	56.84	60.18
W2D	0	37.48	32.88	50.16	50.65
D2D	0	37.46	27.31	44.36	40.56
L2T	30.07	30.27	30.86	30.97	31.08
S2W	27.18	27.37	27.78	28.41	29.24
S2W-Comm	0.0094	0.426	1.03	1.99	3.18
L2T-Comm	0.0045	0.014	0.052	0.187	0.567
W2L	0.014	0.143	0.34	0.471	0.924
Total	**57.28**	**139.4**	**130.7**	**167.36**	**173.42**
Expand Octants	827	6666	53929	433226	3474244
Direct Octants	0	6	126	880	7837
Points	283.7K	2.3M	18.5M	148.9M	1.2B
δ	$\frac{1}{16}$	$\frac{1}{64}$	$\frac{1}{256}$	$\frac{1}{1024}$	$\frac{1}{4096}$

Fig. 7. Isogranular scalability for a gaussian point distribution. For this experiment, we set $\epsilon = 10^{-3}$. The reported times for each component are the maximum values for that component across all the processors. The total wall-clock time is reported in bold face.

for the 1 CPU case. Otherwise, the scalability is very good.

Note that the timings in the nonuniform case have increased considerably compared to the uniform case (Figure 6). A major factor is that the point distribution is random and we cannot use the tensor-product acceleration in the current example. However, there are a few other accelerations that we have not incorporated, we list them below.

- In [9], it is noted that a Hermite expansion is much more effective in condensing the source information than a plane-wave expansion. In the S2W step, significant computational speedup can be achieved by first forming the Hermite expansion and then converting it to a plane-wave expansion using the scheme proposed in [9]. Similary, in the L2T step, instead of directly evaluating the local plane-wave expansion, it is beneficial to first convert the local plane-wave expansion into a local Taylor expansion and then evaluate the Taylor expansion at the targets.
- In this example, we have set $c = 1$. From Figure 7, it is clear that the tree T_d is taking disproportionately more time than T_e. There exists an optimal value for the c depending on the free parameters m, δ and ϵ, below which the timings are higher because of higher cost for the truncation algorithm and above which, the timings are higher because there will be fewer points for FGT box. However, we did not compute the optimal value for this experiment; our choice of c for this experiment was rather arbitrary.

VI. Conclusions

We have presented fast adaptive parallel algorithms to compute discrete spatial transforms with Gaussian-type kernels. We introduced a novel translation scheme that leads to significant speed-ups for non-uniform distributions. We described a new paradigm that uses octrees to compute these transforms on non-uniform distributions with optimal complexity. We demonstrated the good scalability of our implementation for both uniform as well as non-uniform distributions. The code developed as part of this paper will be made publicly available under the terms of the GNU general public license (GPL).

There are a few more accelerations that we can include in our implementation. Most important of these is the Hermite to plane-wave conversion scheme of [9] which is known to yield significant speed-ups. Another important direction is overlapping communication with computation, which can be done for almost all of the communication steps. For example, the plane-wave expansions can be formed at the inter-processor boundaries first and while they are being communicated across processors, plane-wave expansions can be formed at the interior boxes. In addition, we plan to port the local operations to OpenCL [26] to enable the code to run on heterogeneous platforms including multicore CPUs and GPUs.

Apart from the parallel algorithms described in this paper, fast solvers for linear parabolic PDEs require convolutions with

certain non-standard kernels [7], [8]. We are currently investigating fast parallel algorithms for those kernels. Together, we believe, they will provide promising alternatives for applications in high-performance computing.

ACKNOWLEDGMENTS

We would like to thank Leslie Greengard and Marina Spivak for discussions on the fast Gauss transform. R.S. Sampath was supported by Oak Ridge National Laboratory's (ORNL) Postdoctoral Research Associates Program under contract numbers DE-AC05-00OR22725 and DE-AC05-00OR22750. Hari Sundar was supported by Siemens Corporate Research. S. K. Veerapaneni was supported by the National Science Foundation under the grants OCI-0749162 and CNS-0614770.

REFERENCES

[1] J. Strain, "Fast adaptive methods for the free–space heat equation," *SIAM Journal on Scientific Computing*, vol. 15, no. 1, pp. 185–206, 1994.

[2] A. Elgammal, R. Duraiswami, and L. S. Davis, "Efficient kernel density estimation using the fast Gauss transform with applications to color modeling and tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 11, pp. 1499–1504, 2003.

[3] M. Broadie and Y. Yamamoto, "Application of the fast Gauss transform to option pricing," *Management Science*, vol. 49, no. 8, pp. 1071–1088, 2003.

[4] J. Kim, I. Fisher, J.W., A. Yezzi, M. Cetin, and A. Willsky, "A nonparametric statistical method for image segmentation using information theory and curve evolution," *Image Processing, IEEE Transactions on*, vol. 14, no. 10, pp. 1486 –1502, oct. 2005.

[5] S. K. Veerapaneni and G. Biros, "The Chebyshev fast Gauss and nonuniform fast Fourier transforms and their application to the evaluation of distributed heat potentials," *Journal of Computational Physics*, vol. 227, pp. 7768–7790, 2008.

[6] R. Kress, *Linear Integral Equations*, ser. Applied Mathematical Sciences. Springer, 1999.

[7] J.-R. Li and L. Greengard, "High order accurate methods for the evaluation of layer heat potentials," *SIAM Journal on Scientific Computing*, vol. 31, no. 5, pp. 3847–3860, 2009.

[8] S. K. Veerapaneni and G. Biros, "Arbitrary-order accurate schemes for computing boundary heat potentials," *Xxxxx, preprint*, 2009.

[9] M. Spivak, S. K. Veerapaneni, and L. Greengard, "The fast generalized Gauss transform," *(In press) SIAM Journal on Scientific Computing*, 2010.

[10] J. D. Victor and B. W. Knight, "Simultaneously band and space limited functions in two dimensions and receptive fields of visual neurons," in *Perspectives and Problems in Nonlinear Science*, E. Kaplan, J. E. Marsden, and K. R. Sreenivasan, Eds. Springer, 2003, ch. 13.

[11] L. Greengard and J. Strain, "The fast Gauss transform," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 1, pp. 79–94, 1991.

[12] L. Greengard and X. Sun, "A new version of the fast Gauss transform," *Documenta Mathematica*, vol. III, pp. 575–584, 1998.

[13] X. Sun and Y. Bao, "A kronecker product representation of the fast gauss transform," *SIAM J. Matrix Anal. Appl.*, vol. 24, no. 3, pp. 768–786, 2002.

[14] C. Yang, R. Duraiswami, N. Gumerov, and L. S. Davis, "Improved fast Gauss transform and efficient kernel density estimation," *Proceedings of Ninth IEEE International Conference on Computer Vision*, pp. 664–671, 2003.

[15] J. Tausch and A. Weckiewicz, "Multidimensional fast gauss transforms by chebyshev expansions," *SIAM Journal on Scientific Computing*, vol. 31, no. 5, pp. 3547–3565, 2009.

[16] M. Rio Yokota, L. A. Barba and G. Knepley, "Petrbf–a parallel o(n) algorithm for radial basis function interpolation," *arXiv:0909.5413v1*, 2009.

[17] Y. Yamamoto, "Efficient parallel implementation of a weather derivatives pricing algorithm based on the fast gauss transform," in *IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society, 2006.

[18] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc home page," 2001, http://www.mcs.anl.gov/petsc.

[19] J. Kruskal, Joseph B., "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956. [Online]. Available: http://www.jstor.org/stable/2033241

[20] T. Corman, C. Leiserson, and R. Rivest, *Introduction to Algorithms*. MIT Press, 1990.

[21] G. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," IBM Ltd., Tech. Rep., 1966.

[22] H. Sundar, R. S. Sampath, and G. Biros, "Bottom-up construction and 2:1 balance refinement of linear octrees in parallel," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2675 – 2708, 2008.

[23] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, "PETSc users manual," Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 2.1.5, 2004.

[24] R. S. Sampath, S. S. Adavani, H. Sundar, I. Lashuk, and G. Biros, "Dendro: parallel algorithms for multigrid and AMR methods on 2:1 balanced octrees," in *SC '08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. Piscataway, NJ, USA: IEEE Press, 2008, pp. 1 – 12.

[25] NCCS, "Jaguar's system architecture," http://www.nccs.gov/computing-resources/jaguar.

[26] K. Group, "Opencl - the open standard for parallel programming of heterogeneous systems," 2009, http://www.khronos.org/opencl/.