Parallel Fast Gauss Transform

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

***Abstract*—We present a fast parallel algorithm to compute the sum of N Gaussians at N points in an optimal $\mathcal{O}(\frac{N}{n_p})$ time on n_p processors. Our implementation is based on the plane-wave representation of the kernels which permits diagonal translation. We present a new algorithm for translation that minimizes work and memory loads compared to previous schemes. Computing the transform to six-digit accuracy at 120 billion points took 185 seconds on 4096 cores of Jaguar.**

I. Introduction

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

$$F(x_j) = \sum_{k=1}^{N} G_\delta(\|x_j - y_k\|) f_k \quad \text{at} \quad \{x_j \,|\, j = 1, ..., M\}, \quad (1)$$
$$\text{where} \quad x_j, y_k \in \mathbb{R}^d.$$

where the kernel G_δ is smooth, exponentially decaying in both physical and Fourier domain. The parameter δ controls how rapidly the kernel decays. In the Gauss transform case, $G_\delta(\|x_j - y_k\|) = e^{-\frac{\|x_j-y_k\|^2}{\delta}}$. We call the points x as targets and y as sources.

Discrete sums of the form (1) are encountered in a variety of disciplines including computational physics, machine learning, computational finance and computer graphics. Computing these sums directly takes $\mathcal{O}(NM)$ time and is not feasible for large scale problems. Starting from the earlier work of Greengard and Strain [2], several sequential algorithms have been proposed (e.g., [3], [5], [1]) to reduced the cost to an optimal $\mathcal{O}(N + M)$. However, to our knowledge, there have been no parallel implementations to date.

Contributions.

- We propose a new sequential algorithm to compute (1) when the sources are distributed on arbitrarily adaptive grids. Our scheme is an extension of the tree-splitting scheme proposed in [4] for computing continuous Gauss transforms.
- We present a novel sequential and parallel translation scheme for the plane wave expansions. This scheme is particularly minimizes the computational cost and memory requirements when the source distribution is highly nonuniform.
-

II. Overview of FGT

For simplicity, we assume that the points are uniformly distributed and that they reside within a unit cube. The design of fast algorithms for (1) is strongly dependent on three independent parameter viz., 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. If there are fewer targets than a threshold value n^*, we use a simple *truncation algorithm*, otherwise, we use a *expansion algorithm*. The threshold value depends on all three independent parameters and we will discuss its choice after introducing both algorithms.

A. Truncation algorithm

Since the kernel in (1) decays exponentially, we can simply truncate the sum as

$$F(x_j) = \sum_{y_k \in \mathcal{I}[x_j]} G_\delta(\|x_j - y_k\|) f_k \quad (2)$$

where $\mathcal{I}[x_j]$ is the interaction list which includes all the sources that are within a distance $\sqrt{\delta \ln(1/\epsilon)}$. Beyond this distance, a Gaussian centered at x_j decays below ϵ. The complexity of this algorithm is $\mathcal{O}(N\sqrt{\delta \ln(1/\epsilon)})$. This is a common technique used in the graphics community. When δ is large and/or high accuracy is required, the cost of this algorithm grows quadratically.

B. Expansion algorithm

There are two variations of FGT: one based on hermite expansion [2] and another based on plane-wave expansion [3]. The former has lower expansion costs while the latter has lower translation costs. The latter version is further improved in [1] for volumetric data. Our implementation is based on [1] and we summarize it here.

The central is the finite-term plane-wave representation of the kernel,

$$G_\delta(\|x_j - y_k\|) \approx \sum_{|k| \le p} \hat{G}(k) e^{i\lambda k \cdot (x_j - y_k)}, \quad \lambda = \frac{L}{p\sqrt{\delta}} \quad (3)$$

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, we have

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

The algorithm begins by partitioning the domain into uniform boxes of size $\sqrt{\delta}$ each. A Gaussian located at the center of a box B decays below ϵ beyond a fixed number of boxes. We call these boxes the interaction list of B, denoted by $\mathcal{I}[B]$.

In the fast algorithm, a target point x receives information from a source point y via

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

$$w_k = \tag{5}$$

2) W2L: The plane wave expansion of each box is transmitted to all the boxes in its interaction list.

$$v_k = \tag{6}$$

3) L2T: The local plane wave expansion is evaluated at the target locations.

When there are significantly more number of sources in each box, it is easy see why we

C. A novel scheme for translation

Once the wave expansions are formed at all the FGT boxes, the next step is to form each of their local expansions. Since the FGT boxes are of size $\mathcal{O}(\sqrt{\delta})$, only a fixed number of surrounding boxes contribute to the local expansion of a box B. We shall call these set of boxes as its *interaction list*, denoted by $\mathcal{I}[B]$. A *direct scheme* forms the local expansion by simply visiting all the boxes in $\mathcal{I}[B]$ and translating their wave expansions. After initializing the local expansions $\{v_k \quad \forall \quad |k| \leq p\}$ to zero, the pseudo-code for the direct scheme is:

```
DIRECT SCHEME
for each $C \in \mathcal{I}[B]$ do
    $v_k^B \; += e^{iz_k \cdot (c^B - c^C)/\sqrt{\delta}} w_k^C \quad \forall \quad |k| \leq p$
end for
```

Assuming the size of $\mathcal{I}[B]$ is $K^3 - 1$, this algorithm requires $\mathcal{O}(K^3 p^3 N_B)$ work to form local expansions at all the boxes.

a) Accelerating the plane-wave translation step: The sweeping algorithm discussed in [1] reduces this cost to $\mathcal{O}(9p^3 N_B)$. This algorithm is however not memory efficient when there are "holes" in the domain. We propose the following modification which has a much smaller memory footprint.

- First compute the local expansions at the outermost layer of the FGT boxes. These are shown in Figure 1 in orange. It is possible to speed up this initial computation as well since we can compute the local expansions for a small number of boxes and use the propagation rule.
- We propagate the local expansions from this initial layer to subsequent layers. The main task is to understand this propagation. Consider the case shown in Figure 3 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 (in Orange).
- The local expansion $v_k^{B(i+1,j+1)}$ can be written in terms for the local expansions of $B(i,j), B(i+1,j)$ and $B(i,j+1)$, along with the corner box which is in the influence list of $B(i+1,j+1)$ but not of the others $(+)$ and the box which is in the influence list of $B(i,j)$ but not of the other three$(-)$. This is marked in Figure 3 assuming $K = 3, n = 1$. The local expansion is given by,

$$v_k^{i+1,j+1} = e^{iz_k s/\sqrt{\delta}} v_k^{i+1,j} + e^{iz_k s/\sqrt{\delta}} v_k^{i,j+1} - e^{iz_k s/\sqrt{\delta}} v_k^{i,j} - e^{iz_k ns/\sqrt{\delta}}$$

- The propagation can then be used to propagate to the remaining boxes in the new propagation layer. At any given stage only the values of the propagation layer needs to be stored. The values of any non-zero FGT boxes doesn't need to be remembered.

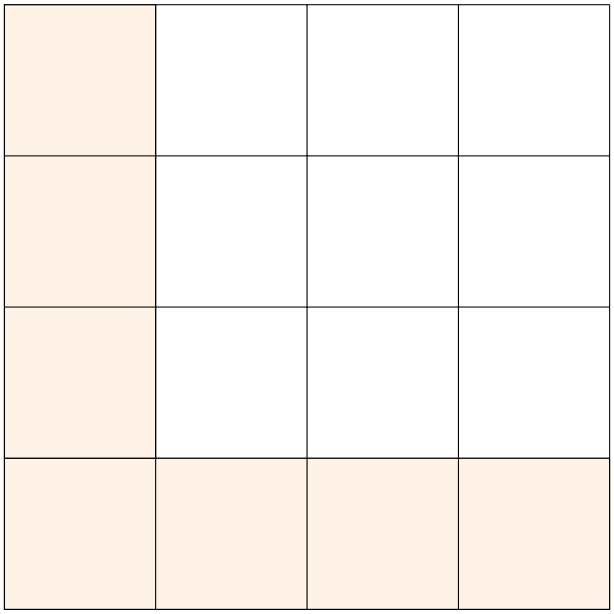


Fig. 1. The outermost layer, for which the local expansions are computed directly.

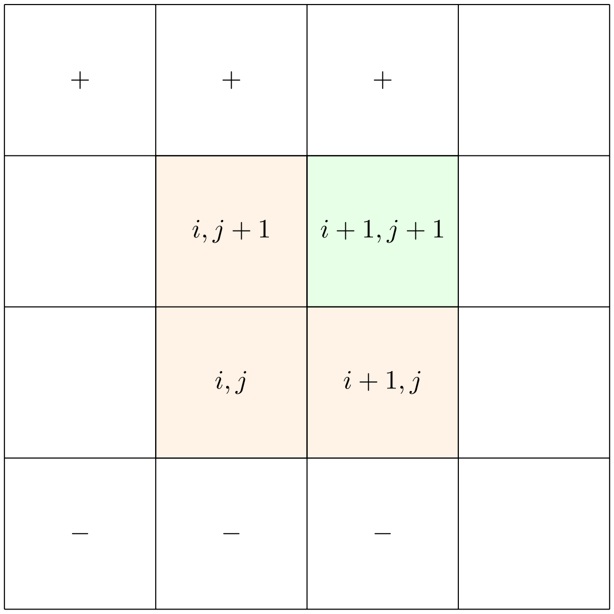


Fig. 2. The Contributions.

Based on the cost of the expansion based algorithm, we

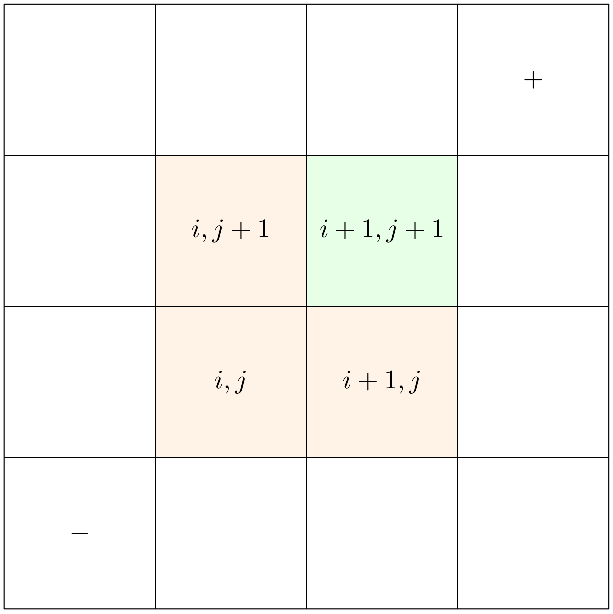


Fig. 3. The propagation of the local expansions using neighbors.

```
if $N\sqrt{\delta \ln\left(\frac{1}{\epsilon}\right)} < \frac{1}{2}(2p)^3$ then
  Use truncation algorithm
else
  Use expansion based algorithm
end if
```

III. Nonuniform distributions

MOTIVATION

In the uniform distribution case, we can precisely estimate the threshold n_{th} that decides whether one should use the truncation algorithm or the expansion based algorithm. However, when the source and target distributions are highly nonuniform, as is the case in most practical applications, it is not straightforward. For example, when we superimpose a regular grid structure of FGT on a nonuniform distribution, some boxes will have lot of points while some are almost empty.

OCTREES We assume sources and targets are the same for simplicity. We also assume that the octree is constructed so that there are no more than a fixed number of points in each leaf node.

Algorithm 1 *Tree Splitting*

```
for each leaf node $\ell$ do
  if $|\ell| > \sqrt{\delta}$ then
    assign $\ell$ to $T_d$ (direct or truncation
    based)
  else
    assign $\ell$ to $T_e$ (expansion based)
  end if
end for
```

Algorithm 2 *FGT on a split tree*

```
for each $\ell \in T_e$ do
  (S2W) Add contribution of point in $\ell$ to
  the FGT box it belongs
end for

(W2L) Form local expansions using
sweeping

EVALUATE THE EFFECT OF ALL SOURCES IN $T_d$
for each $\ell \in T_d$ do
  for $x \in \ell$ do
    Add contribution of $x$ to all the
    target boxes (in $T_e$) and target points
    (in $T_d$)
  end for
end for

EVALUATE THE EFFECT OF ALL SOURCES IN $T_e$
for each FGT box $B \in T_e$ do
  Use local expansion for target points
  within $B$

  Use wave expansion for target points in
  $T_e$ that are within its interaction list
end for
```

[In the parallel case, the interaction between T_d and T_e can be done while processors are communicating other info.]

IV. Results

Operation	Max. Time	Avg. Time	Max. Flops	Avg. Flops
S2W				
W2L				
L2T				
Total				

TABLE I

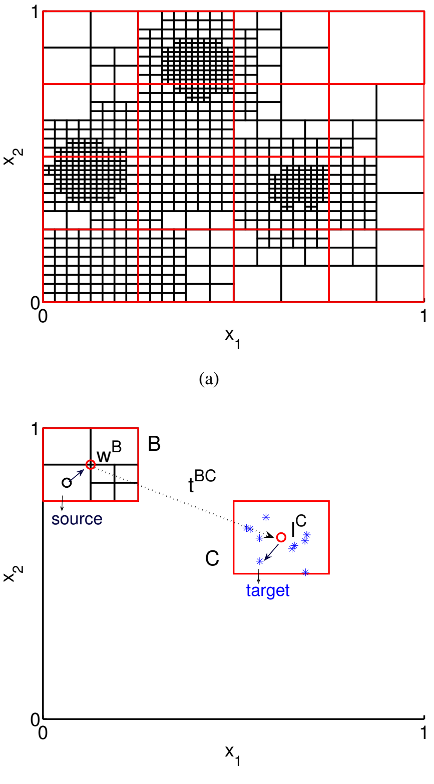
Timings on 37,268 (32^3) processors on Jaguar. The point distribution is uniform random, the parameter $\delta = 8 \times 10^{-4}$ and precision $\epsilon = 10^{-6}$. Each processor has a million points.

V. Conclusions

Several acceleration techniques for forming and evaluating plane wave expansion were introduced in [1]. We will incorporate these in our final submission. When the sources come from a tensor product grid in each octant, the expansion constants can be reduced from exponential to linear in dimension. In the sequential case, this speed-up will make the cost of the algorithm comparable to that of fast Fourier transform (FFT). Since the parallel performance of FFT has been sub-optimal till date, we believe that our parallel FGT would be the method of choice even for regular grids.

Fig. 6. Strong scaling on Jaguar. The problem size is fixed to 1 billion points and the parameters are the same as defined in Table IV. [Rahul: Try varying the num. of processors from 16 to atleast 8192]

Fig. 7. Weak scaling on Jaguar. Number of points per proceesor is fixed at 1 million (so $N = 10^6 \times n_p$) and the parameter $\delta = \frac{10}{N^{1/3}}$. [Rahul: Try varying the num. of processors from 16 to atleast 8192]



(a)

(b)

Fig. 4. *Here we illustrate one of the main steps involved in FGT. a) The leaf nodes (shown in black) of the given quadtree are assigned to regular boxes (shown in red). The size of the boxes is chosen so that each box corresponds to a node at some level of the quadtree. b) At a source box B, the influence of all the constituent leaf nodes is encoded in the wave expansion $\mathbf{w}^B$. The translation operator $\mathbf{t}^{BC}$ transfers the wave expansion of B to the target box C. At C, the influence of all the source boxes in $\mathcal{I}[C]$ is encoded in the local expansion $\mathbf{l}^C$. Finally, $G_\delta f$ at a target that belongs to C is given by $< \mathbf{l}^C, \mathbf{v}^C(\mathbf{x}) >$.*

A few assumptions to simplify our implementation:

1) *Each processor has finite number of FGT boxes. If we also care for the contrary, we would have a box that spans across processors and hence S2W and L2T also need to parallelized. Not a big deal, but simplifies our job for now.*
2) *We will assume that $\delta = 2^{-n}$ for some even number n and FGT box size $h = \sqrt{\delta}$. This will help us in reducing book-keeping: otherwise, in the octree case, we will have FGT boxes cutting across leaf nodes.*

References

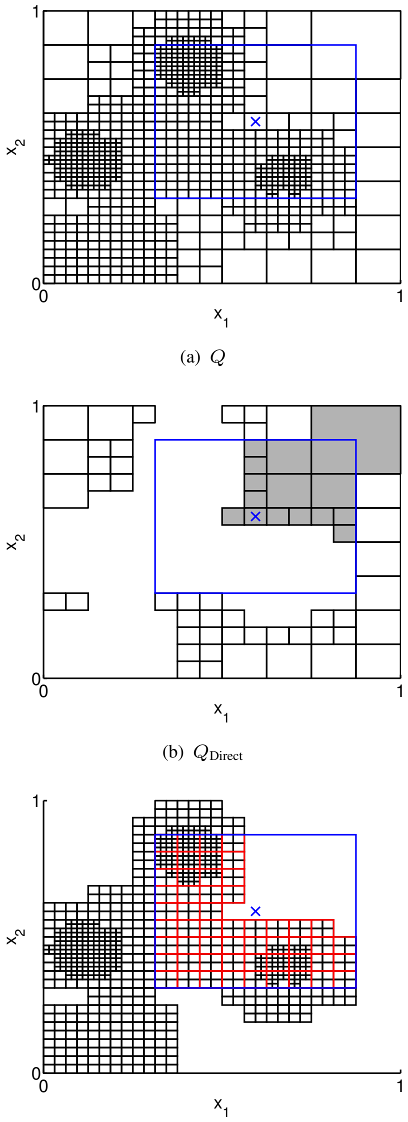
[1] *The fast generalized Gauss transform*, Xxxx, (2010).

[2] L. Greengard and J. Strain, *The fast Gauss transform*, SIAM Journal on Scientific and Statistical Computing, 12 (1991), pp. 79–94.

[3] Leslie Greengard and Xiaobai Sun, *A new version of the fast Gauss transform*, Documenta Mathematica, III (1998), pp. 575–584.

[4] Shravan K. Veerapaneni and George Biros, *The Chebyshev fast Gauss and nonuniform fast fourier transforms and their application to the evaluation of distributed heat potentials*, Journal of Computational Physics, 227 (2008), pp. 7768–7790.

[5] Changjiang Yang, Ramani Duraiswami, Nail Gumerov, and Larry S. Davis, *Improved fast Gauss transform and efficient kernel density estimation*, Proceedings of Ninth IEEE International Conference on Computer Vision, (2003), pp. 664–671.



(a) Q

(b) Q_{Direct}

(c) Q_{Expand}

Fig. 5. *In this figure we illustrate the tree splitting that we use in CFGT to obtain an algorithmic complexity that is independent of the value of* δ*. Subfigure (a) shows the quadtree, a target* $\mathbf{x}$*, and the support of the Gaussian centered at* $\mathbf{x}$ *(blue box). For this* δ*, we can observe that a direct evaluation would be expensive because we have to loop through all the leaf nodes within the support. A kernel expansion based approach would also be expensive because it generates too many boxes. Hence, for optimal complexity, we split the tree into two parts. Subfigure (b) shows the tree with the size of leaf nodes being greater than* r_b*. To evaluate the Gauss transform at a point* $\mathbf{x}$*, we just need to visit the leaf nodes in the support of the Gaussian (the shaded ones). There can be a maximum of* $(2n+1)^2$ *of such leaf nodes. Subfigure (c) show the tree with the size of leaf nodes being smaller or equal than* r_b*. In this tree we use the kernel expansion-based algorithm. Instead of visiting all the leaf nodes within the support, we only have to gather the wave expansion of the source boxes (red boxes) that are within the support. The number of these source boxes is bounded from above by* $(2n+1)^2$.