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 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. Computing these sums directly on a single CPU would take $\mathcal{O}(N^2)$ time and is not feasible for large scale problems. 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 use our parallel octree implementation (Sundar et al. SIAM J. SCI. COMPUT. Vol. 30, No. 5, 2008) to efficiently handle non-uniform distributions. We incorporate a plane-wave representation of the Gaussian kernel which permits diagonal translation. We introduce a novel scheme for translating the plane-waves to reduce the computation and storage costs in the case of nonuniform distributions. Computing the transform to six-digit accuracy at 120 billion points took 185 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 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

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

$$\text{where} \quad x_j, y_k \in \mathbb{R}^d.$$

The kernel G_δ is a smooth exponentially decaying function in both the physical and Fourier domains. 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. If the kernel decays rapidly, one can simply truncate the sum to a few neighboring sources at each target. However, in many important applications, this is not the case and computing these sums directly takes $\mathcal{O}(NM)$ time.

Starting from the earlier work of Greengard and Strain [1], several sequential algorithms have been proposed (e.g., [2], [6], [4], [3]) to reduce the cost to an optimal $\mathcal{O}(N+M)$. However, to our knowledge, there have been no parallel implementations to date.

a) Contributions: Our a The main contributions of this work are given below.

- We present a first ever parallel implementation of the fast gauss transform for an uniform distribution of points. We incorporate the accelerations introduced in [3] for tensor product grids. Thereby, the complexity constants in our scheme scale linearly with the number of dimensions as opposed to exponential growth.
- We present a novel scheme for the translation of plane wave expansions; this is one of the steps in the sequential fast gauss transform algorithm. This new scheme reduces the computation and storage costs compared to the previous implementations, especially for highly non-uniform point distributions.
- We extend our schemes to nonuniform distributions. OCTREES.
- The cost FGT grows for smaller values of δ. In the nonuniform case, We present a scheme that scales well in all ranges of the parameters. It is an extension of the tree-splitting scheme proposed in [5] for computing continuous Gauss transforms.

The rest of the paper is organized as follows. In Section II, we give a high-level description of the sequential FGT algorithm for uniform point distributions. We introduce a novel translation scheme in Section III. In the uniform distribution case, our scheme is slightly (nearly twice) more expensive compared to sweeping algorithm introduced in [2]. In the nonuniform case, our scheme significantly improves the storage and computational costs. We also discuss its parallel implementation. In Section IV, we will present our sequential and parallel algorithms for nonuniform distributions. Finally, in Section V, we will present scalability results of our algorithm using a variety of test cases.

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 to

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

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

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

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

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

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

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 $\{v_k \;\; \forall \;\; |k| \le p\}$ to zero, the pseudo-code for the direct scheme is:

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

III. Translation scheme

b) Accelerating the plane-wave translation step: The sweeping algorithm discussed in [3] 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.

Based on the cost of the expansion based algorithm, we

```
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
```

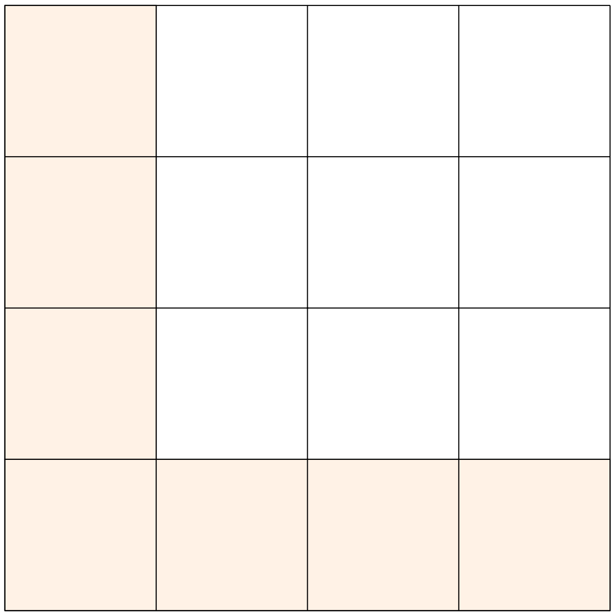

Fig. 1. The outermost layer, for which the local expansions are computed directly.

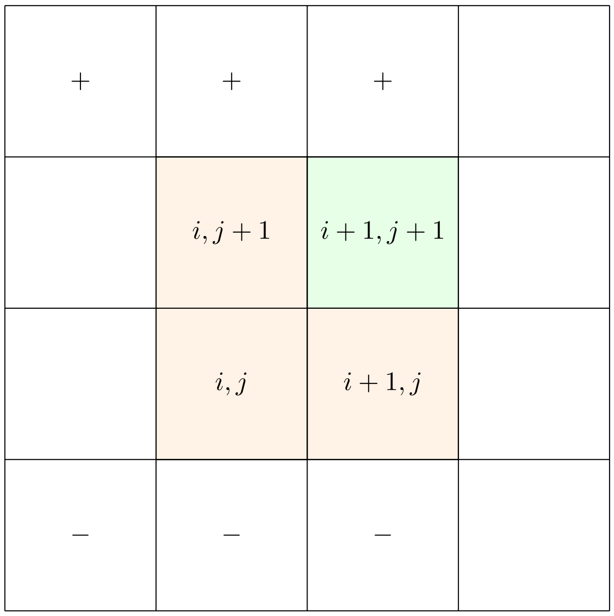

Fig. 2. The Contributions.

IV. 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.

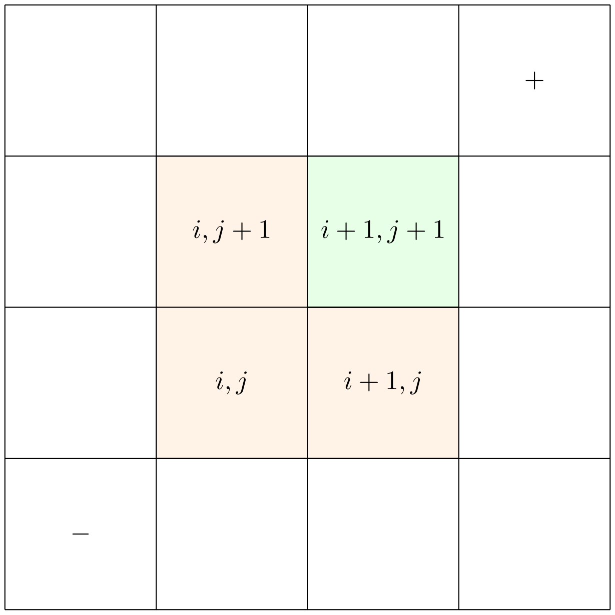

Fig. 3. The propagation of the local expansions using neighbors.

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
```

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

V. 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.

VI. Conclusions

Several acceleration techniques for forming and evaluating plane wave expansion were introduced in [3]. We will incorporate these in our final submission. When the sources come from

Fig. 4. Strong scaling on Jaguar. The problem size is fixed to 1 billion points and the parameters are the same as defined in Table V. [Rahul: Try varying the num. of processors from 16 to atleast 8192]

Fig. 5. 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]

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
```

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.

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.*

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) Post-doctoral Research Associates Program under contract numbers DE-AC05-00OR22725 and DE-AC05-00OR22750. S. Veerapaneni was supported by the National Science Foundation under the grants OCI-0749162 and CNS-0614770.