A DISTRIBUTED KERNEL SUMMATION FRAMEWORK FOR MACHINE LEARNING AND SCIENTIFIC APPLICATIONS

A Thesis
Presented to
The Academic Faculty

by

Dong Ryeol Lee

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
Computer Science

Georgia Institute of Technology
August 2012

A DISTRIBUTED KERNEL SUMMATION FRAMEWORK FOR MACHINE LEARNING AND SCIENTIFIC APPLICATIONS

Approved by:

Professor Alexander G. Gray, Advisor
School of Computational Science and Engineering
Georgia Institute of Technology

Professor Edmond Chow
School of Computational Science and Engineering
Georgia Institute of Technology

Professor Christos Faloutsos
School of Computer Science
Carnegie Mellon University

Professor Haesun Park
School of Computational Science and Engineering
Georgia Institute of Technology

Professor Richard Vuduc
School of Computational Science and Engineering
Georgia Institute of Technology

Date Approved: May 4, 2012

To my parents and my sister for their support and patience...

ACKNOWLEDGEMENTS

I would like to thank my advisor Alex Gray for his guidance and support. I started working with Alex eight years ago as an undergraduate researcher at Carnegie Mellon University. He introduced me to the problem of nearest neighbor, and I spent the entire summer of 2004 working with another undergraduate in doing one of the most comprehensive literature and empirical survey of different nearest neighbor methods. I gained a deeper understanding of data structures and algorithms through this project and subsequently decided to follow Alex to Georgia Tech for continuing my Ph.D. studies. Alex believed in my hidden strengths and abilities as a researcher, and this provided me with the necessary motivation and energy to finish the long journey of Ph.D. dissertation research.

I would also like to thank my committee member, Professor Rich Vuduc, for his support. Without his help and advice, I could not have had the chance to turn my dissertation research around when all hope seemed to be lost. I would also like to thank Professor Haesun Park and Professor Edmond Chow for agreeing to serve on the committee. Professor Park's course on numerical linear algebra was immensely helpful and laid the foundation for gaining appreciation of their applications in fast data analysis methods. With Professor Edmond Chow, I am currently involved in an interesting hybrid data analysis and HPC project with several other students in the CSE department. Last but not least, I would like to thank Professor Christos Faloutsos for serving on the dissertation committee on a very short notice. I am grateful to him for taking his valuable time and providing immensely helpful comments on improving the presentation materials. I also thank anonymous reviewers who provided corrections to a part of my thesis draft and interviewers who helped

me hone my research focus.

I have become a more mature and better person after meeting great people at Georgia Tech. No words can express my thanks and appreciations for people who have befriended me. I still remember the conversation (both casual and research-related) we had while eating food or drinking coffee at Student Center, Moe's, Tin Drum, Spoon, and other places at Tech Square. Engaging in physical activities at the Campus Recreation Center was helpful in building up stamina and endurance for research. Although many have already moved onto new places and I will be moving on as well, I believe our paths will cross again.

I would like to thank my parents and my sister for their patience during my graduate studies. My thesis took longer than expected yet they have been always available for providing moral support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The class of computational problems I consider in this thesis share the common trait of requiring consideration of pairs (or higher-order tuples) of data points. For problems modeling pairwise interactions, we consider accelerating the operations on N by N matrices of the form: $K = k(x_i, x_j)_{i,j}$ where $k(\cdot, \cdot)$ is the function that outputs a real value given x_i and x_j from the data set. I focus on the problem of kernel summation operations ubiquitous in many data mining and scientific algorithms.

In machine learning, kernel summations appear in popular kernel methods which can model nonlinear structures in data. Kernel methods include many non-parametric methods such as kernel density estimation, kernel regression, Gaussian process regression, kernel PCA, and kernel support vector machines (SVM). In computational physics, kernel summations occur inside the classical N-body problem for simulating positions of a set of celestial bodies or atoms.

This thesis attempts to marry, for the first time, the best relevant techniques in parallel computing, where kernel summations are in low dimensions, with the best general-dimension algorithms from the machine learning literature. We provide a unified, efficient parallel kernel summation framework that can utilize:

1. Various types of deterministic and probabilistic approximations that may be suitable for both low and high-dimensional problems with a large number of data points.

2. Indexing the data using any multi-dimensional binary tree with both distributed memory (MPI) and shared memory (OpenMP/Intel TBB) parallelism.

3. A dynamic load balancing scheme to adjust work imbalances during the computation.

I will first summarize my previous research in serial kernel summation algorithms. This work started from Greengard/Rokhlin's earlier work on fast multipole methods for the purpose of approximating potential sums of many particles. The contributions of this part of this thesis include the followings: (1) reinterpretation of Greengard/Rokhlin's work for the computer science community; (2) the extension of the algorithms to use a larger class of approximation strategies, i.e. probabilistic error bounds via Monte Carlo techniques; (3) the multibody series expansion: the generalization of the theory of fast multipole methods to handle interactions of more than two entities; (4) the first $\mathcal{O}(N)$ proof of the batch approximate kernel summation using a notion of intrinsic dimensionality. Then I move onto the problem of parallelization of the kernel summations and tackling the scaling of two other kernel methods, Gaussian process regression (kernel matrix inversion) and kernel PCA (kernel matrix eigendecomposition).

The artifact of this thesis has contributed to an open-source machine learning package called MLPACK which has been first demonstrated at the NIPS 2008 and subsequently at the NIPS 2011 Big Learning Workshop. Completing a portion of this thesis involved utilization of high performance computing resource at XSEDE (eXtreme Science and Engineering Discovery Environment) and NERSC (National Energy Research Scientific Computing Center).

CHAPTER I

INTRODUCTION

This thesis focuses on scaling key bottleneck inner-loop computations on distributed datasets. Data may be distributed because: 1) it is more cost-effective to distribute data on a network of less powerful nodes than storing everything on one powerful node; 2) it allows distributed query processing for high scalability. Each process (which may/may not be on the same node) owns a subset of data and needs to initiate communications (i.e. MPI, memory-mapped files) when it needs a remote piece of data owned by another process. Many machine learning and scientific simulations require running the computations on multiple parameter settings. Especially, the number of points can be prohibitively large so that one CPU cannot handle the computation in a tractable amount of time. Unlike the usual three-dimensional setting in N-body simulations, D may be as high as or more than 1000 in many machine learning methods. My thesis attempts to provide a general framework that encompasses acceleration techniques for a wide range of both low-dimensional and high-dimensional problems with a large number of data points.

1.1 What This Thesis is About

The class of computational problems I consider in my thesis share the common trait of requiring consideration of pairs (or higher-order tuples) of data points. For problems modeling pairwise interactions, we consider accelerating the operations on N by N matrices of the form: $\mathbf{K} = \{k(\mathbf{x_i}, \mathbf{x_j})\}_{i,j}$ where $k(\cdot, \cdot)$ is the *function* that outputs a real value given $\mathbf{x_i}$ and $\mathbf{x_j}$ from the data set. I focus on the following three *fundamental linear algebraic operations* ubiquitous in many data mining and scientific algorithms:

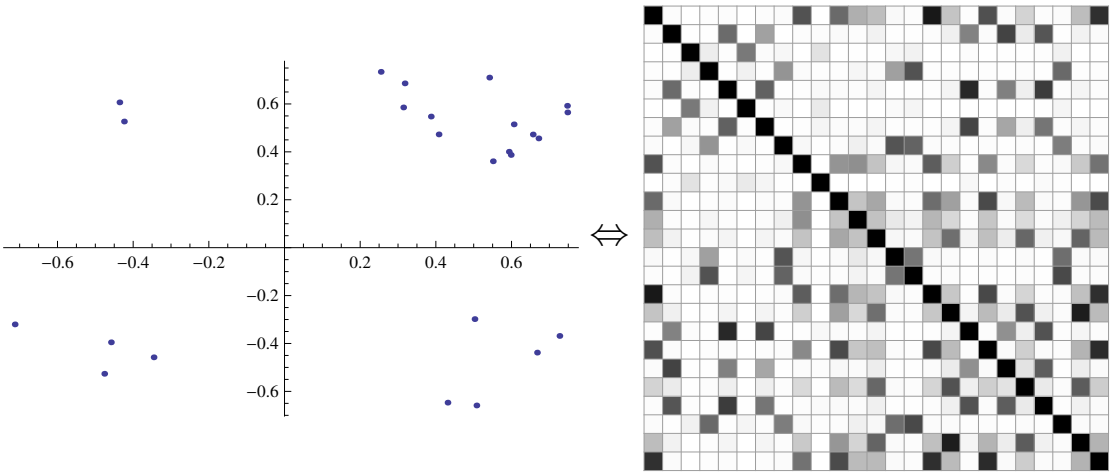


Figure 1: Left: an example two-dimensional point set. **Right:** the kernel matrix formed by the point set shown in the left using the Gaussian kernel $k(x, y) = e^{\frac{-||x-y||^2}{2h^2}}$.

1. Kernel summations: $\forall \mathbf{x_1} \in \mathbf{X_1}, \sum_{\mathbf{x_2} \in \mathbf{X_2}} k(\mathbf{x_1}, \mathbf{x_2})$
2. Solving a linear system involving a kernel matrix: $\mathbf{K}^{-1}\mathbf{y}$
3. Eigendecomposing matrices: $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$

In machine learning, the three operations above appear in widely used *kernel methods* which can model nonlinear structures in data and include many non-parametric methods such as kernel density estimation [145], kernel regression [138], Gaussian process regression [153], kernel PCA [164], and kernel support vector machines (SVM) [165]. In computational physics, kernel summations occur inside the classical N-body problem for simulating positions of a set of celestial bodies or atoms.

There are two main issues that one needs to address for developing scaling the computations above on a distributed setting. First, acceleration is generally feasible only by trading accuracy for speed. Therefore, it is very crucial for researchers and scientists to be able to both quantify and control approximation errors. Secondly, achieving scalability in a distributed setting requires additional considerations such as: 1) minimizing inherently serial portions of the algorithm (Amdahl's law); 2) minimizing the time spent in critical sections; 3) overlapping communication and

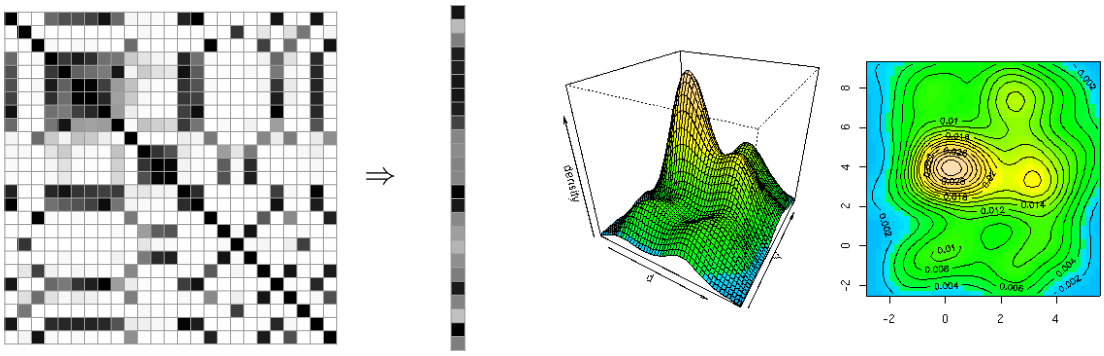


Figure 2: If the kernel is a probability density function, then we can do density estimation. Kernel density estimation [145] involves the computation of weighted column average of the kernel matrix $\sum_{\mathbf{r_i} \in \mathbf{R}} w_i k(\mathbf{q}, \mathbf{r_i}) = \mathbf{Kw}$.

computation as much as possible. In my thesis, I utilize 1) OpenMP for shared-memory parallelism and 2) MPI for distributed-memory parallelism; it is also possible to utilize (optionally) the CUDA programming framework for harnessing massive parallelism in General Purpose Graphics Processing Units (GPGPUs) available on today's commodity machines but we will not explore it in this thesis.

The overall framework is derived in [117]. My thesis attempts to marry, for the first time, the best relevant techniques in parallel computing, where kernel summations are in low dimensions, with the best general-dimension algorithms from the machine learning literature.

Thesis Statement: "Utilizing the best general-dimension algorithms, approximation methods with error bounds, the distributed and shared memory parallelism can help scale kernel methods."

In this thesis, I provide a unified, efficient parallel kernel summation framework that can utilize:

1. Various types of deterministic and probabilistic approximations that may be suitable for both low and high-dimensional problems with a large number of

data points.

2. Indexing the data using any multi-dimensional binary tree with both distributed memory (MPI) and shared memory (OpenMP) parallelism.

3. A dynamic load balancing scheme to adjust work imbalances during the computation.

The framework provides a general approach for accelerating the computation of many popular machine learning methods. The motivation is similar to that of [119] and [98]. In [119], a general framework was developed to support various types of scientific simulations. In [98], several graph mining operations (PageRank, Random Walk with Restart (RWR), diameter estimation, and connected components) was parallelized via an implementation of *Generalized Iterated Matrix-Vector multiplication* (GIM-V) on HADOOP platform [21]. This thesis is based on parallelization of the previously successful *generalized N-body framework* [78, 128] which is similar to the well-known spatial join algorithms [64, 26] and is an extension of the parallelization work in [25].

The techniques analyzed and developed in this thesis have wide applications in performing efficient, accurate computation in molecular dynamics, statistical modeling, and astrophysical simulations. I utilize various techniques from numerical optimizations, approximation theory, computational geometry, and high-performance computing to develop scalable implementations. The components of the overall framework in my thesis are the followings:

Reinterpretation of Greengard and Rokhlin's Work: Greengard and Rokhlin's seminar work on the *Fast Multipole Methods* provided the foundation for kernel summation methods using hierarchical data structures. However, the theorem-proof presentation of their proposed approximation schemes is hard for non-experts to understand. In my thesis, I re-cast Greengard and Rokhlin's derivations in terms of the

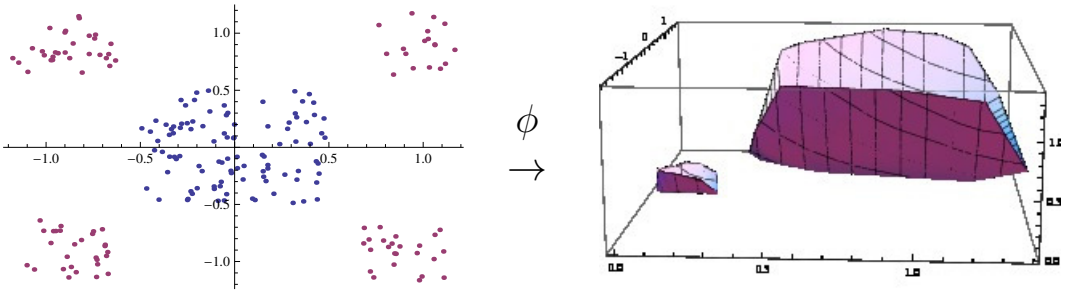


Figure 3: A kernel k is a similarity function. If it in addition satisfies the Mercer's conditions ($\mathbf{K} \succ 0$), then it corresponds to a dot-product: $k(\mathbf{x_i}, \mathbf{x_j}) = \phi(\mathbf{x_i})^T \phi(\mathbf{x_j})$.

more general framework called *generalized N-body framework* [78].

High-dimensional Kernel Summations: Classical approaches of accelerating kernel sums using hierarchical data structures do not scale to higher-dimensions. Due to the *curse of dimensionality*, distances between all pairs of points in high dimensions concentrate around one value and hierarchical data structures such as *kd*-trees and metric-trees adapt poorly to underlying data. High-dimensional kernel summations need to be dealt with in kernel principal component analysis and kernel support vector machines.

None of the previous approaches for kernel summations addresses the issue of reducing the computational cost of each distance computation which incurs $\mathcal{O}(D)$ cost. However, the *intrinsic dimensionality* d of most high-dimensional datasets is much smaller than the explicit dimension D (that is, $d \ll D$). In my thesis, I provide two approaches for handling high-dimensional kernel summations. First, I extend the original fast multipole-type methods to use approximation schemes with both hard and probabilistic error. Second, I propose a new data structure called subspace tree which maps each data point in the node to its lower dimensional mapping as determined by any linear dimension reduction method such as PCA. This new data structure is suitable for reducing the cost of each pairwise distance computation, the

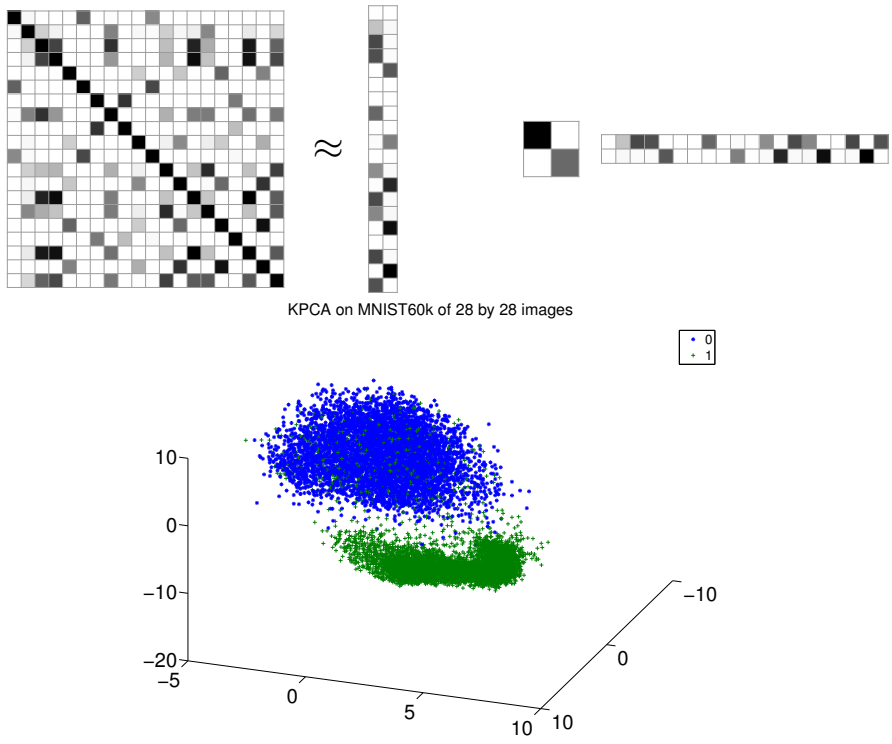


Figure 4: Top: Eigendecomposition of a kernel matrix into a set of kernel eigenvectors and kernel eigenvalues. **Bottom:** Projection of the MNIST [111] training dataset onto its first two kernel principal components.

most dominant cost in many kernel methods.

Higher-order Generalization of Fast Multipole Methods (Chapter 7): I generalize the theory of *fast multipole methods* to handle interactions of more than two entities, so-called *multibody series expansion*. A three-body potential function can account for interactions among triples of particles which are uncaptured by pairwise interaction functions such as Coulombic or Lennard-Jones potentials. Likewise, a multibody potential of order n can account for interactions among n-tuples of particles uncaptured by interaction functions of lower orders. To date, the computation of multibody potential functions for a large number of particles has not been possible

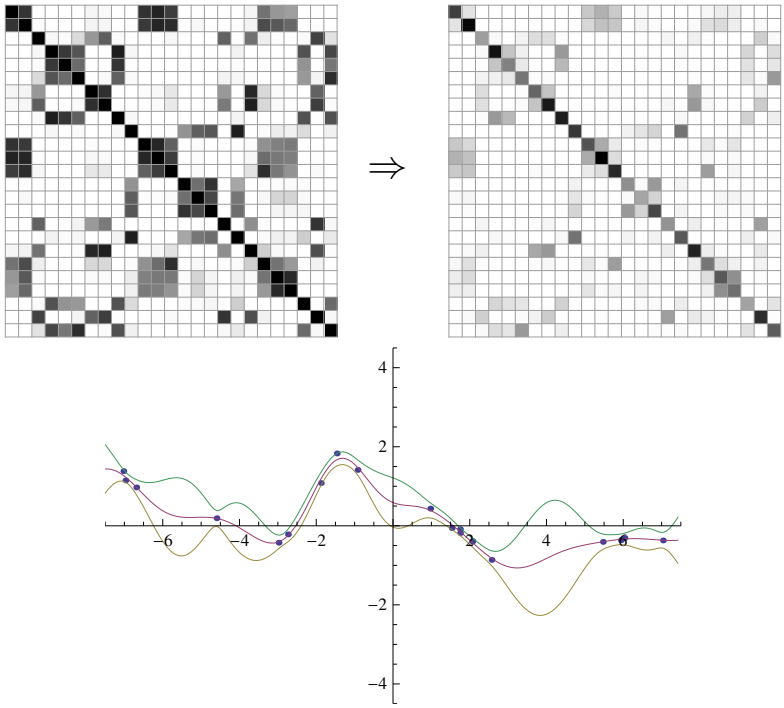


Figure 5: Inverting the kernel matrix $\mathbf{K}^{-1}$. An example application includes the well-known Gaussian process regression.

due to its $\mathcal{O}(N^n)$ scaling cost.

$$\Phi(\mathbf{x}; \underbrace{\mathbf{X} \times \cdots \times \mathbf{X}}_{(n-1) \text{ copies}}) = \sum_{\mathbf{x_{i_2}} \in \mathbf{X} \setminus \{\mathbf{x}\}} \sum_{\substack{\mathbf{x_{i_3}} \in \mathbf{X} \setminus \{\mathbf{x}\} \\ i_2 < i_3}} \cdots \sum_{\substack{\mathbf{x_{i_n}} \in \mathbf{X} \setminus \{\mathbf{x}\} \\ i_{n-1} < i_n}} k(\mathbf{x}, \mathbf{x_{i_2}} \cdots, \mathbf{x_{i_n}}) \tag{1.1.1}$$

For the first time, I provide a fast Barnes-Hut type algorithm for efficiently approximating multibody potentials. My approach guarantees a user-specified bound on the absolute or relative error in the computed potential. I provide speedup results on a three-body dispersion potential, the Axilrod-Teller potential [9]. This work is in submission to a journal in computational physics community [116].

Runtime analysis of Kernel Summations: Previous runtime analysis of the pairwise kernel summation method used the assumption of uniformity of the data distribution. I use a new new notion of distribution-dependent measure called the *expansion constant*, which has been successfully used in proving the $\mathcal{O}(\log N)$ runtime

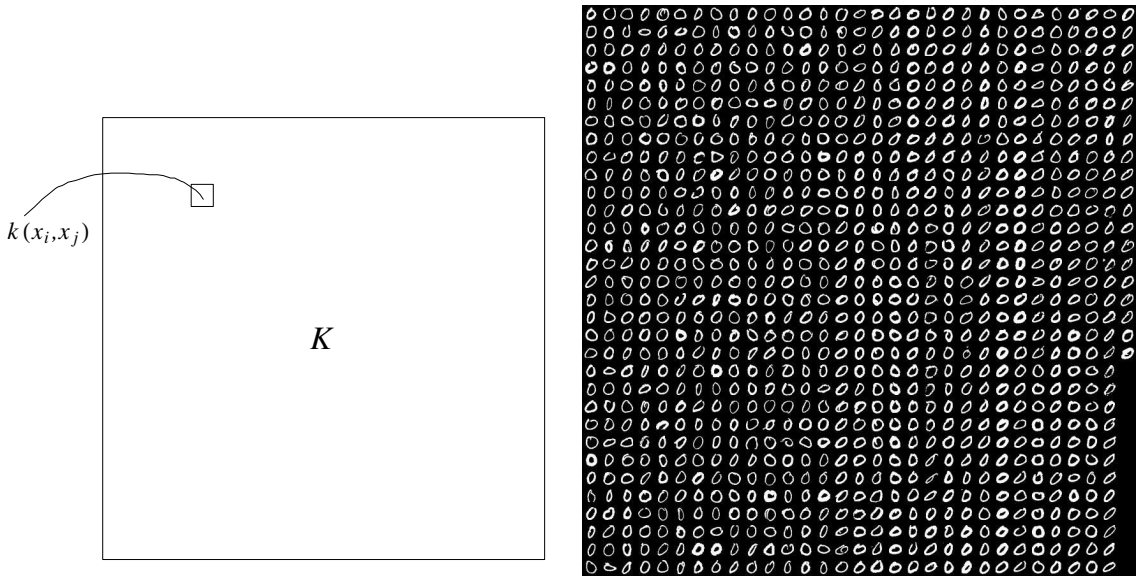


Figure 6: Storing the kernel matrix in the main memory for even a small number of points is expensive. For example, the MNIST [111] handwritten digit recognition data of sixty-thousand 28×28 images requires over 28 GB to store the kernel matrix in double precision.

of the nearest neighbor search of a single query point [20]. I provide the first rigorous proof of the linear time complexity of the approximate kernel summation. This is not included in the thesis but published in [151].

1.2 Open-source Machine Learning: MLPACK

I have been involved in an open-source development of machine learning package called **MLPACK** aimed for large-scale data analysis. As both a numerical software developer and researcher, I believe that writing and sharing robust code is imperative for the advancement of science. For example, well-documented source code for *Fast Multipole Methods* is hard to find. Therefore, I have made my C++ implementation available as a part of **MLPACK** for other researchers. Maintaining **MLPACK** required utilizing many open source software for scientific computing, such as Trilinos [93], Armadillo linear algebra library [163], the GNU Scientific Library [49] and Boost Library [107]. **MLPACK** has been first demonstrated at the NIPS 2008 [24] and subsequently at the NIPS 2011 Big Learning Workshop [53]. I have utilized

high performance computing resources such as XSEDE (eXtreme Science and Engineering Discovery Environment) and NERSC (National Energy Research Scientific Computing Center) for completing a portion of this thesis.

1.3 Structure of This Thesis and Notations

This thesis is organized into the following parts:

- Chapter 2 first defines the general class of kernel summation problems and introduces preliminary materials on data structures, algorithmic framework, and approximation methods.

- Chapter 3 elucidates the mathematical machinery underneath the popular fast multipole methods using figures and algorithms. As a running example, it derives for the first time a hierarchical version of the famous fast Gauss transform using the $\mathcal{O}(p^D)$ Cartesian expansion.

- Chapter 4 derives another hierarchical version of the fast Gauss transform using the $\mathcal{O}(D^p)$ Cartesian expansion.

- Chapter 5 proposes two strategies for scaling kernel summations to higher dimensions using a Monte Carlo sampling and a data structure that is able to capture dominant subspace information.

- Chapter 6 provides an application of fast pairwise kernel summations to mean shift, a popular nonparametric clustering. Nonparametric clustering is then used for the task of image segmentation.

- Chapter 7 for the first time extends the pairwise kernel summation framework to a higher-order kernel summation. Based on this new derivations, this chapter provides a fast hierarchical algorithm for the problem of multibody potential

summation for accurately modeling higher-order interactions in molecular dynamics.

- Chapter 8 provides the parallel extension for kernel summations. It introduces the method for building a large-scale distributed multidimensional tree and the distributed kernel summation framework.

- Chapter 9 introduces the distributed averaging framework. It then shows how to combine it with the random feature extraction method to create fast kernel eigendecomposition and kernel inversion methods.

- Chapter 10 summarizes the contributions and outlines some possible future research directions.

Throughout this thesis, we use these common sets of notations:

- **(Normal Distribution).** This is denoted by $\mathcal{N}(\mu, \boldsymbol{\Sigma})$.

- **(Matrix, Vector, Scalar).** We denote a matrix by a bold uppercase alphabet (e.g. $\mathbf{A}$) and a vector by a bold lowercase alphabet (e.g. $\mathbf{a}$). A scalar is always an italicized lowercase or uppercase alphabet (e.g. a or A). Likewise, we denote a function by its output. For example, for a function whose output is a matrix, we denote it by a bold uppercase alphabet (e.g. $\mathbf{F} : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^a \times \mathbb{R}^b$). For a function outputting a vector, $\mathbf{f} : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. Lastly, a function outputting a scalar, $f : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$.

- **(Vector Component).** For a given vector $\mathbf{v} \in \mathbb{R}^k$, we access its d-th component by $\mathbf{v}[d]$ where $1 \leq d \leq k$ (i.e. 1-based index).

- **(Multi-index Notation).** Throughout this paper, we will be using the multi-index notation. A D-dimensional multi-index $\boldsymbol{\alpha}$ is a D-tuple of non-negative integers and will be denoted using a bold lowercase Greek alphabet. For any D-dimensional multi-indices $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and any $\mathbf{x} \in \mathbb{R}^D$,

$$|\boldsymbol{\alpha}| = \boldsymbol{\alpha}[1] + \boldsymbol{\alpha}[2] + \cdots + \boldsymbol{\alpha}[D]$$

$$\boldsymbol{\alpha}! = (\boldsymbol{\alpha}[1])!(\boldsymbol{\alpha}[2])! \cdots (\boldsymbol{\alpha}[D])!$$

$$\mathbf{x}^{\boldsymbol{\alpha}} = (\mathbf{x}[1])^{\boldsymbol{\alpha}[1]}(\mathbf{x}[2])^{\boldsymbol{\alpha}[2]} \cdots (\mathbf{x}[D])^{\boldsymbol{\alpha}[D]}$$

$$D^{\boldsymbol{\alpha}} = \partial_1^{\boldsymbol{\alpha}[1]} \partial_2^{\boldsymbol{\alpha}[2]} \cdots \partial_D^{\boldsymbol{\alpha}[D]}$$

$$\boldsymbol{\alpha} + \boldsymbol{\beta} = (\boldsymbol{\alpha}[1] + \boldsymbol{\beta}[1], \cdots, \boldsymbol{\alpha}[D] + \boldsymbol{\beta}[D])$$

$$\boldsymbol{\alpha} - \boldsymbol{\beta} = (\boldsymbol{\alpha}[1] - \boldsymbol{\beta}[1], \cdots, \boldsymbol{\alpha}[D] - \boldsymbol{\beta}[D]) \text{ for } \boldsymbol{\alpha} \geq \boldsymbol{\beta}.$$

where ∂_i is a i-th directional partial derivative. Define $\boldsymbol{\alpha} > \boldsymbol{\beta}$ if $\boldsymbol{\alpha}[d] > \boldsymbol{\beta}[d]$, and $\boldsymbol{\alpha} \geq p$ for $p \in \mathbb{Z}^+ \cup \{0\}$ if $\boldsymbol{\alpha}[d] \geq p$ for $1 \leq d \leq D$ (and similarly for $\boldsymbol{\alpha} \leq p$).

- **(Set of Points).** Each of these is denoted by a bold upper case alphabet. For a multidimensional dataset, this can be represented as a matrix $\mathbf{P} = \{\mathbf{p_1}, \cdots, \mathbf{p_N}\}$ where each $\mathbf{p_i} \in \mathbb{R}^D$ is a column vector of $\mathbf{P}$.

- **(Distance between a Pair of Points).** $||\cdot||$ is assumed to be the Euclidean metric unless specified otherwise.

- **(Size of a Point Set).** Given a set $\mathbf{S}$, it size is denoted by $|\mathbf{S}|$.

- **(Dimensionality).** We reserve the italicized uppercase D for the dimensionality of the problem.

- **(Probability Guarantee).** We use the unbold Greek alphabet α.

- **(Reference Set/Query Set vs Training Set/Test Set vs Source Set/Target Set).** Following [78, 80, 77], we use the terms *reference set* and *training set* and *source set* interchangably. Likewise, the terms *query set* and *test set* and *target set* have the same meaning. The *query set* is denoted as $\mathbf{Q}$ and the *reference set* is denoted as $\mathbf{R}$.

- **(Subset of a Point Set).** Given a point set $\mathbf{D}$, denote any of its subset by $\mathbf{D}^{sub} \subset \mathbf{D}$. $\mathbf{D}^{sub}$ contains a subset of the columns of $\mathbf{D}$.

- **(A Tree Node).** A tree node represents a subset of a point set represented by the root node. Hence, we use the same notation as the previous.

- **(Representative Point of a Tree Node).** Usually a geometric center is used but any point inside the bounding primitive of a tree node is chosen as well. For the tree node $\mathbf{P}$, this is denoted as $\mathbf{c_P}$.

- **(Child Nodes of an Internal Tree Node).** Given a node $\mathbf{N}$, denote its left and right child nodes by $\mathbf{N}^L$ and $\mathbf{N}^R$ respectively.

CHAPTER II

PRELIMINARY MATERIALS

We first define a general class of problems called *generalized N-body problems* [78].

Definition 2.0.1. *A generalized N-body problem is a computational problem over a set of point sets* $\mathbf{X_1} \cdots, \mathbf{X_n}$ *and associative, commutative operators* $\bigotimes_1, \cdots, \bigotimes_n$ *and* $f : \mathbf{X_1} \times \cdots \times \mathbf{X_n} \to \mathbb{R}$*:*

$$\psi(\mathbf{X_1}, \ldots, \mathbf{X_n}) = \bigotimes_{\mathbf{x_1} \in \mathbf{X_1}}^{1} \cdots \bigotimes_{\mathbf{x_n} \in \mathbf{X_n}}^{n} f(\mathbf{x_1}, \ldots, \mathbf{x_n}) \tag{2.0.1}$$

subject to decomposability requirement for $1 \leq i \leq n$*,*

$$\psi(\ldots, \mathbf{X_i}, \ldots) = \psi(\ldots, \mathbf{X_i}^L, \ldots) \otimes_i \psi(\ldots, \mathbf{X_i}^R, \ldots). \tag{2.0.2}$$

A generalized N-body problem is monochromatic if $\mathbf{X_1} = \cdots = \mathbf{X_n}$*. Otherwise it is a multi-chromatic problem.*

We note that the associative, commutative properties of the operators $\bigotimes_1, \cdots, \bigotimes_n$ let us re-order computations efficiently without changing the results (up to numerical precision) and the decomposability property lets us decompose the problem using a spatial decomposition of each point set $\mathbf{X_1}, \cdots, \mathbf{X_n}$.

In this thesis, we restrict ourselves to a subset of this rather large class of problems called *kernel summation problems*. We start by defining the following useful operator:

Definition 2.0.2. *A map operator is an operator over a given set* $\mathbf{X}$ *and outputs a vector whose component corresponds to each element* $\mathbf{x_i} \in \mathbf{X}$*. That is:*

$$\underset{\mathbf{x_i} \in \mathbf{X}}{\mathbf{map}} f(\mathbf{x_i}, \mathbf{Y}) = \begin{bmatrix} f(\mathbf{x_1}, \mathbf{Y}) \\ \vdots \\ f(\mathbf{x}_{|\mathbf{X}|}, \mathbf{Y}) \end{bmatrix} \tag{2.0.3}$$

We are now ready to define the general class of kernel summation problems[1].

Definition 2.0.3. *A kernel summation problem is a computational problem over a set of point sets* $\mathbf{X_1} \cdots, \mathbf{X_n}$ *and* $k : \mathbf{X_1} \times \cdots \times \mathbf{X_n} \to \mathbb{R}$*:*

$$\mathbf{\Phi}(\mathbf{X_1} \times \cdots \times \mathbf{X_n}) = \mathop{\mathbf{map}}_{\mathbf{x_{i_1}} \in \mathbf{X_1}} \sum_{\mathbf{x_{i_2}} \in \mathbf{X_2}} \cdots \sum_{\mathbf{x_{i_n}} \in \mathbf{X_n}} k(\mathbf{x_{i_1}}, \cdots, \mathbf{x_{i_n}}) \tag{2.0.4}$$

Denote each scalar component of $\mathbf{\Phi}(\mathbf{X_1} \times \cdots \times \mathbf{X_n})$ *by fixing the argument corresponding to the first set* $\mathbf{X_1}$*:* $\Phi(\mathbf{x_1}; \mathbf{X_2} \times \cdots \times \mathbf{X_n})$*.*

The specific kernel summation problems we deal with in this thesis include:

1. Pairwise kernel summations (Chapter 3 and Chapter 5):

$$\mathbf{\Phi}\left(\mathbf{Q} \times \mathbf{R}\right) = \mathop{\mathbf{map}}_{\mathbf{q_i} \in \mathbf{Q}} \left(\sum_{\mathbf{r_j} \in \mathbf{R}} w_j k(\mathbf{q_i}, \mathbf{r_j}) \right) \tag{2.0.5}$$

2. Multibody potential summations (Chapter 7):

$$\mathbf{\Phi}(\underbrace{\mathbf{X} \times \cdots \times \mathbf{X}}_{n \text{ copies}}) = \mathop{\mathbf{map}}_{\mathbf{x} \in \mathbf{X}} \sum_{\mathbf{x_{i_2}} \in \mathbf{X} \setminus \{\mathbf{x}\}} \sum_{\substack{\mathbf{x_{i_3}} \in \mathbf{X} \setminus \{\mathbf{x}\} \\ i_2 < i_3}} \cdots \sum_{\substack{\mathbf{x_{i_n}} \in \mathbf{X} \setminus \{\mathbf{x}\} \\ i_{n-1} < i_n}} k(\mathbf{x}, \mathbf{x_{i_2}} \cdots, \mathbf{x_{i_n}})$$

Later we show that the problems of kernel matrix eigendecomposition and inversion can be reduced to the problem of kernel summations (see Chapter 9).

2.1 Multidimensional Trees

In this thesis, we focus on hierarchical methods because: 1) it is a natural framework to control approximation in a varying degree of resolution; 2) the specialized acceleration techniques can always be used as a base case. We utilize a hierarchical data structure called multidimensional tree to form hierarchical groupings of points based on their locations. We use a variant of *kd*-trees [15] using the recursive procedure shown in Algorithm 2.2.1. Initially, the algorithm starts with $\mathbf{P} = \mathbf{X}$ (the entire point

[1]Note that [94] defines another general class of nested summation problems.

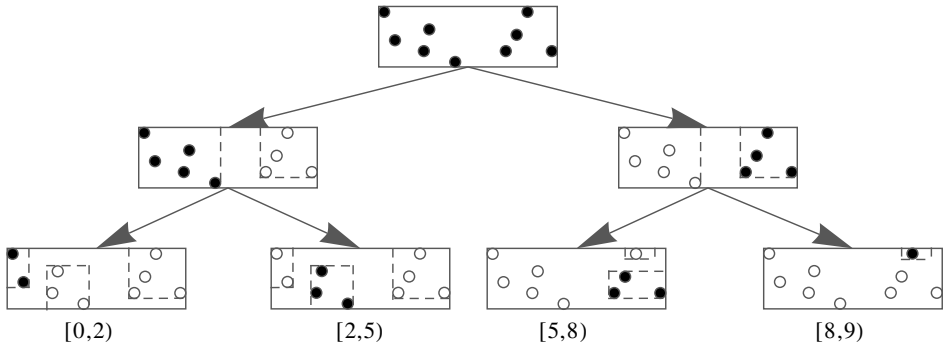


Figure 7: *kd*-tree of a two-dimensional point set. At each level, the bounding box is split in half along the widest dimension. The solid points denote the points owned by each node. At each leaf node, we can enumerate each point with its depth-first rank. The minimum depth-first rank (inclusive) and the maximum depth-first rank (exclusive) is shown for each node.

set). We split a given set of points along the widest dimension of the bounding hyper-rectangle into two equal halves at the splitting coordinate such that the resulting two subsets $\mathbf{P}^L$ and $\mathbf{P}^R$ are disjoint[2] and $\mathbf{P} = \mathbf{P}^L \cup \mathbf{P}^R$. We continue splitting until the number of points is below some user-defined threshold called the *leaf threshold*. If the number of points owned by a node exceeds the leaf threshold, then it is called an *internal node*. Otherwise it is called a *leaf node*. Assuming that each split on a level results in the equal number of points on the left subset and the right subset $\mathbf{P}^L$ and $\mathbf{P}^R$ respectively, the runtime cost is $\mathcal{O}(|\mathbf{X}| \log |\mathbf{X}|)$. We note that the cost of building a *kd*-tree is negligible compared to the computations we perform using it. Other multidimensional trees include octrees, *kd*-trees [15], and cover-trees [20]. Note that Algorithm 2.2.1 is a serial algorithm and we will discuss the parallel construction algorithm in Chapter 8.

Cached Sufficient Statistics. In addition to the bound information, each node is decorated with statistics about the points underneath it. These are called *cached sufficient statistics* [133], and some examples include:

[2]Spill-trees [121] do not obey this property and have been used for speeding up high-dimensional nearest neighbor searches.

Algorithm 2.2.1 BUILDKDTREE($\mathbf{P}$): builds a *kd*-tree from $\mathbf{P}$ (see Figure 7).

if $|\mathbf{P}|$ is above the leaf threshold **then**
  Find the widest dimension d of the bounding box of $\mathbf{P}$.
  Choose an axis-aligned split s along d.
  Split $\mathbf{P} = \mathbf{P}^L \cup \mathbf{P}^R$ where $\mathbf{P}^L = \{\mathbf{x} \in \mathbf{P} \mid \mathbf{x}[d] \leq s\}$ and $\mathbf{P}^R = \mathbf{P} \backslash \mathbf{P}^L$.
  BUILDKDTREE($\mathbf{P}^L$), BUILDKDTREE($\mathbf{P}^R$)
  Form far-field moments of $\mathbf{P}$ by translating far-field moments of $\mathbf{P}^L$ and $\mathbf{P}^R$.
else
  Form far-field moments of $\mathbf{P}$.
Initialize summary statistics of $\mathbf{P}$.

1. Covariance, principal eigendirections and eigenvectors of the owned points.

2. Far-field moments in fast multipole methods and their variants [78, 80, 82, 77, 83, 84, 85, 86].

We can efficiently perform these statistics on each leaf node and recursively *translating* the statistics of the children node for an internal node. See Figure 8 and Section 3.1.3.

2.2 Generalized N-body Framework

The general framework for computing Equation (2.0.1) is formalized in [78, 82, 80, 77]. This approach consists of the following steps:

1. Build a spatial tree (such as *kd*-trees) for each of the particle sets $\mathbf{X_1}, \cdots, \mathbf{X_n}$ and their cached sufficient statistics **(Bottom-up phase)**[3]

2. Perform a *multi-tree traversal* over n-tuples of nodes **(Approximation phase)**.

3. Pre-order traverse the tree and propagate unincorporated bound changes downward **(Top-down phase)**.

The generalized N-body framework using the multi-tree traversal is shown in Algorithm 2.2.2, (called by setting each $\mathbf{P_i} = \mathbf{X}$ for $1 \leq i \leq n$), a recursive function

[3]It is possible to build a single tree on the union of the particle sets $\mathbf{X_1}, \cdots, \mathbf{X_n}$ (as done in the FMM literature) and apply the same framework, but we do not consider this approach since the extension is trivial.

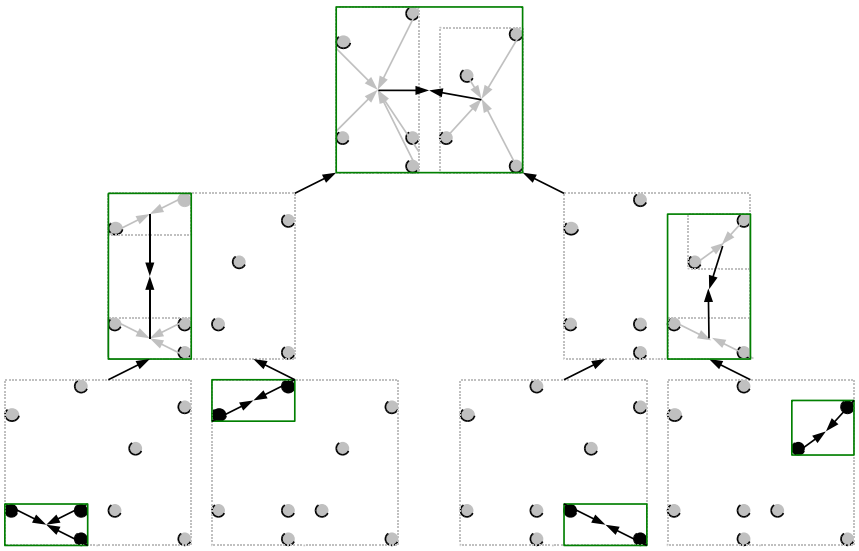


Figure 8: Efficient bottom-up translation of cached sufficient statistics. This includes the far-to-far translation operator in fast multipole method literature.

that allows us to consider the n-tuples formed by choosing each $\mathbf{x_i}$ from $\mathbf{P_i}$; we can gain efficiency over the naive enumeration of the n-tuples by using the bounding box and the moment information stored in each $\mathbf{P_i}$. One such information is the distance bound computed using the bounding box (see Figure 10).

CanSummarize function tests whether $\mathbf{\Phi}(\mathbf{X_1}, \ldots, \mathbf{X_n})$ can be approximated within the error tolerance determined by the algorithm. If the approximation is not possible, then the algorithm continues to consider the data at a finer granularity; it chooses an internal node $\mathbf{P_k}$ (typically the one with the largest diameter) to split among $\{\mathbf{P_i}\}_{i=1}^{n}$. Before recursing to two sub-calls in Line 9 and Line 10 of Algorithm 2.2.2, the algorithm can optionally push quantities from a node that is being split to its child nodes (see Line 8 and Figure 16). After returning from the recursive calls, the node that was just split can refine *summary statistics* based on the results accumulated on its child nodes (see Line 11 and Figure 17).

The basic idea is to terminate the recursion as soon as possible, i.e. by considering a tuple of large subsets and avoiding the number of exhaustive leaf-leaf-leaf

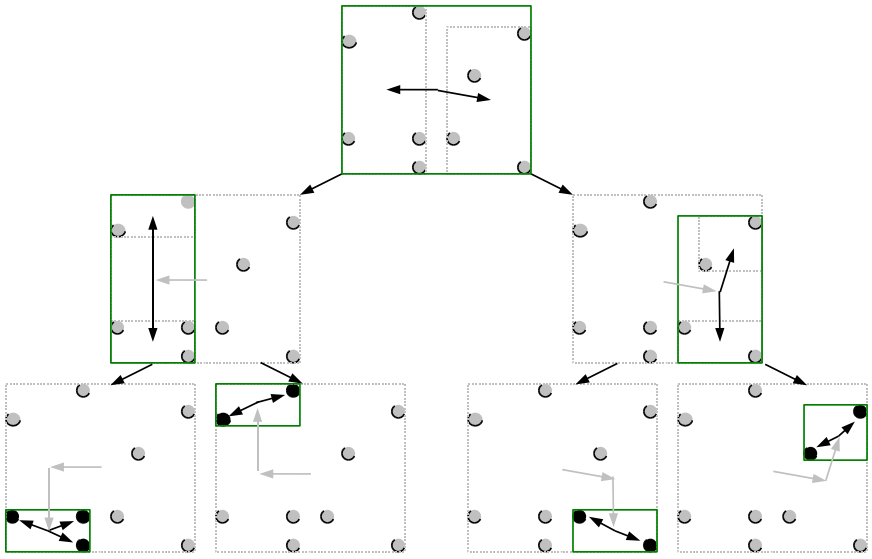


Figure 9: Efficient tow-down propagation. This includes the local-to-local translation operator in fast multipole method literature.

computations. We note that the CanSummarize and Summarize functions effectively replace unwieldy interaction lists used in FMM algorithms. Interaction lists in n-tuple interaction, if naively enumerated, can be large depending on the function k and the dimensionality D of the problem, whereas the generalized N-body approach can handle a wide spectrum of problems without this drawback.

2.3 Approximation Methods

We first focus on the pairwise kernel summation form ($n = 2$) whose canonical form is given in Equation 2.0.5. The computation tree for a pairwise problem is shown in Figure 11. For a given level, some computations are independent; some have to be computed sequentially. The key question is how to specify the CanSummarize function so that the recursive call terminates at a higher pair $\mathbf{Q}^{sub} \times \mathbf{R}^{sub}$.

The key idea is to exploit the properties of the function k and compress the point sets $\mathbf{Q}^{sub}$ and $\mathbf{R}^{sub}$. These are roughly divided into the followings:

1. k has a **finite extent**: a kernel is zero outside a certain domain. An example would be the Epanechnikov kernel $k(\mathbf{q}, \mathbf{r}) = \max\left\{0, 1 - \frac{\|\mathbf{q}-\mathbf{r}\|^2}{h^2}\right\}$.

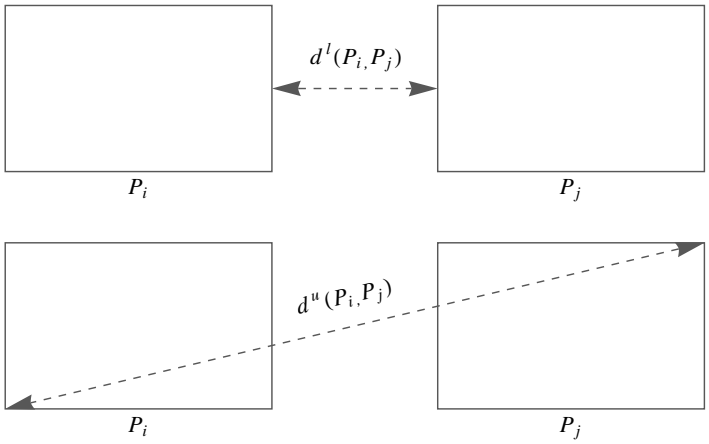


Figure 10: The lower and upper bound on pairwise distances between the points contained in a pair of nodes.

Algorithm 2.2.2 MULTITREE($\{\mathbf{P_i}\}_{i=1}^n$): The canonical multi-tree algorithm.

```
if CANSUMMARIZE({P_i}_{i=1}^n)   (Try approximation.) then
  SUMMARIZE({P_i}_{i=1}^n)
else
  if all of S_i are leaves then
    MULTITREEBASE({P_i}_{i=1}^n)   (Base case.)
  else
    Find an internal node P_k to split among {P_i}_{i=1}^n.
    Propagate bounds of P_k to P_k^L and P_k^R.
    MULTITREE({P_1, ..., P_{k-1}, P_k^L, P_{k+1}, ..., P_n})
    MULTITREE({P_1, ..., P_{k-1}, P_k^R, P_{k+1}, ..., P_n})
    Refine summary statistics based on the two recursive calls.
```

2. k is **numerically low-rank** given the training set: the kernel matrix $\mathbf{K}$ has a low rank implying the data points lie in a span of a smaller subset in the induced reproducing kernel Hilbert space.

3. k is **(conditionally) positive-definite**.

Definition 2.3.1. Positive definite kernel: *Let χ be a nonempty set. If $k : \chi \times \chi$ for all $m \in \mathbb{N}$ and all $\mathbf{x_1}, \cdots, \mathbf{x_m} \in \chi$ gives rise to a positive definite kernel matrix K, then k is positive-definite. That is, $\sum_{i,j} c_i \bar{c}_j \mathbf{K}_{i,j} \geq 0$ for all $c_i \in \mathbb{C}$.*

Conditionally positive definite kernel: *k is conditionally positive definite if $\sum_{i,j} c_i \bar{c}_j \mathbf{K}_{i,j} \geq 0$ for all $\sum_{i=1}^m c_i = 0$.*

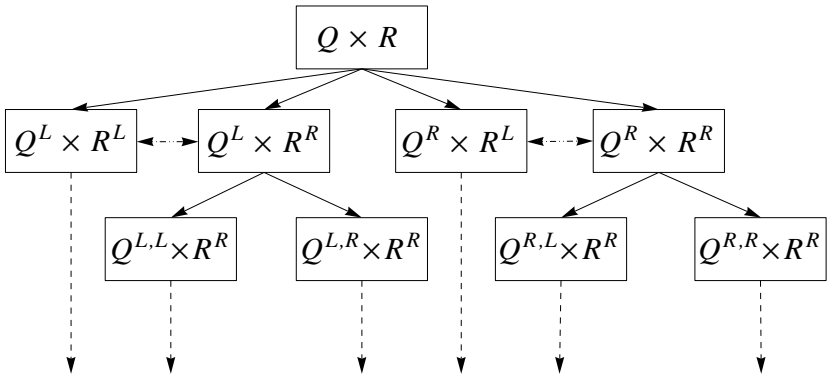


Figure 11: The dual-tree algorithm is given in Algorithm 2.3.1 and is a special case of Algorithm 2.2.2. An example computation tree for a pairwise N-body problem. Here we branch simultaneously both of the sets Q and R. Vertical dashed arrows denote independent computations. Each point in Q is associated with a query result which must be synchronized under parallelization setting. Horizontal dashed arrows denote computations that must be performed sequentially due to this reason.

Fast algorithms for evaluating kernel sums can be divided into four types: 1) hierarchical methods which employ spatial partitioning structures. These methods achieve efficiency via series expansion or finite-extent nature of the kernel; 2) reduced set methods from physics/machine learning communities [165]; 3) Monte Carlo-based methods in the data space; 4) random projection-based methods in the function space.

2.3.1 Hierarchical Methods

Series Expansion-based Methods. Most hierarchical methods using trees utilize series expansions. The first expansion called the far-field expansion summarizes the contribution of $\mathbf{R}^{sub}$ for a query $\mathbf{q}$:

$$\Phi(\mathbf{q}; \mathbf{R}^{sub}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} k(\mathbf{q}, \mathbf{r_{j_n}}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \sum_{m=1}^{\infty} b_m \eta_m(\mathbf{q}, \mathbf{R}^{sub}) \psi_m(\mathbf{r_{j_n}}, \mathbf{R}^{sub})$$

$$= \sum_{m=1}^{\infty} \eta_m(\mathbf{q}, \mathbf{R}^{sub}) \left(\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} b_m w_{j_n} \psi_m(\mathbf{r_{j_n}}, \mathbf{R}^{sub}) \right) = \sum_{m=1}^{\infty} \eta_m(\mathbf{q}, \mathbf{R}^{sub}) M_m(\mathbf{R}^{sub})$$

where η_m's and ψ_m's show dependence on the subset $\mathbf{R}^{sub}$. The second type called

Algorithm 2.3.1 DUALTREE($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$): The canonical dual-tree algorithm.

```
if CANSUMMARIZE(Q^sub, R^sub)   (Try approximation.) then
  SUMMARIZE(Q^sub, R^sub)
else
  if Q^sub is a leaf node then
    if R^sub is a leaf node then
      DUALTREEBASE(Q^sub, R^sub)   (Base case.)
  else
    Propagate bounds of Q^sub to Q^sub,L and Q^sub,R.
    if R^sub is a leaf node then
      DUALTREE(Q^sub,L, R^sub), DUALTREE(Q^sub,R, R^sub)
    else
      DUALTREE(Q^sub,L, R^sub,L), DUALTREE(Q^sub,R, R^sub,L)
      DUALTREE(Q^sub,L, R^sub,R), DUALTREE(Q^sub,R, R^sub,R)
    Refine summary statistics based on the recursive calls.
```

the local expansion for $\mathbf{q} \in \mathbf{Q}^{sub} \subset \mathbf{Q}$ expresses the contribution of $\mathbf{R}^{sub}$ near $\mathbf{q}$:

$$\Phi(\mathbf{q}; \mathbf{R}^{sub}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} k(\mathbf{q}, \mathbf{r_{j_n}}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \sum_{m=1}^{\infty} g_m \eta_m(\mathbf{r_{j_n}}, \mathbf{Q}^{sub}) \psi_m(\mathbf{q}, \mathbf{Q}^{sub})$$

$$= \sum_{m=1}^{\infty} \psi_m(\mathbf{q}, \mathbf{Q}^{sub}) \left(\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} g_m w_{j_n} \eta_m(\mathbf{r_{j_n}}, \mathbf{Q}^{sub}) \right) = \sum_{m=1}^{\infty} \psi_m(\mathbf{q}, \mathbf{Q}^{sub}) L_m(\mathbf{R}, \mathbf{Q}^{sub})$$

Both representation are truncated at a finite number of terms depending on the level of prescribed accuracy, achieving $\mathcal{O}(|\mathbf{Q}| \log |\mathbf{R}|)$ runtime in most cases. To achieve $\mathcal{O}(|\mathbf{Q}| + |\mathbf{R}|)$ runtime, we require an efficient linear operator that converts $M_m(\mathbf{R})$ into $L_m(\mathbf{R}, \mathbf{Q})$'s. Depending on the basis representations of η's and ψ's, the far-to-local linear operator is diagonal and the translation is linear in the number of coefficients. There are many serial algorithms [7, 10, 83, 84, 30, 78, 203] that use different series expansions forms to bound error deterministically.

Finite-extent Kernels. While most FMM algorithms have focused on approximating continuous differentiable kernel sums, [78, 80, 82, 77] have generalized to handle any kernels including the popular Epanechnikov kernel widely used in statistics. A region of space outside the finite extent from a given query point (or a group of query points) are pruned using the distance criterion. [156] has also generalized and

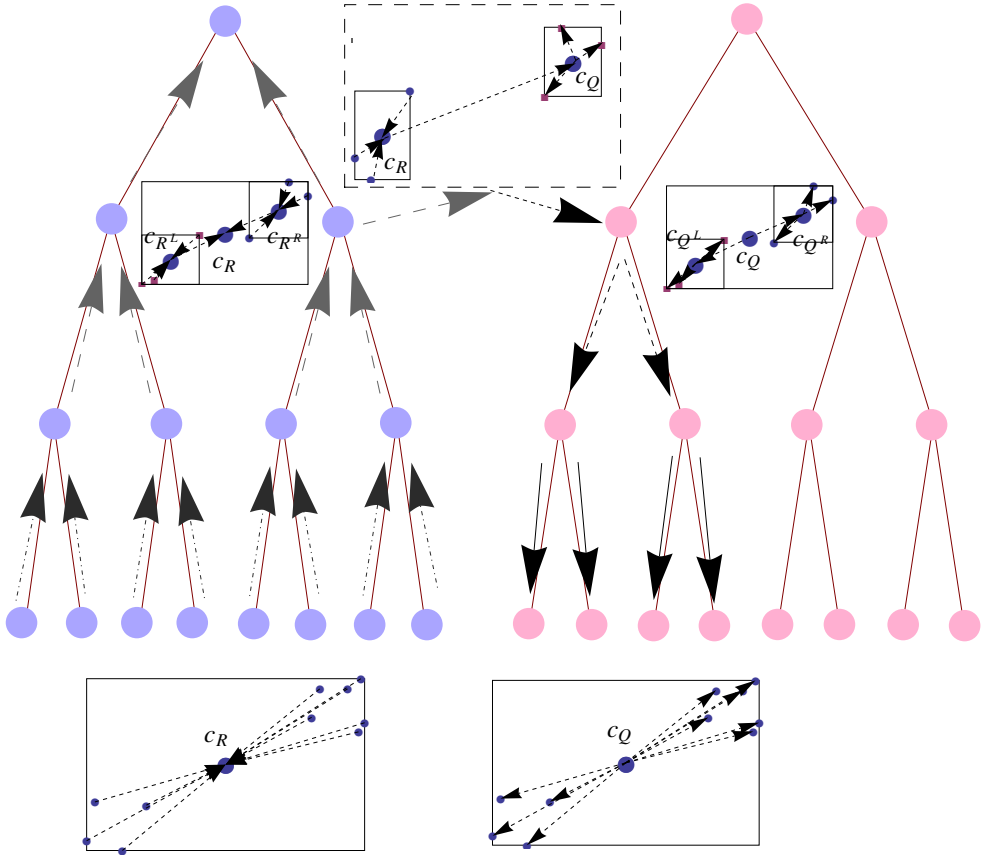


Figure 12: The reference points (the left tree) are hierarchically compressed and uncompressed when a pair of query (from the right tree)/reference nodes is approximated within an error tolerance.

developed a series-expansion-like moments for the Epanechnikov kernel.

2.3.2 Reduced Set Expansion-based Methods

Reduced set methods express each data point as a linear combination of points (so called dictionary points each of which gives rise to the function $b : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$):

$$\Phi(\mathbf{q}; \mathbf{R}) \approx \widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{reduced}) = \sum_{\mathbf{d_k} \in \mathbb{R}^D} u_k b(\mathbf{q}, \mathbf{d_k})$$

where $|\mathbf{R}^{reduced}| \ll |\mathbf{R}|$ and the resulting kernel sum can be evaluated more quickly. In the physics community, uniform grid points are chosen and points are projected on Fourier bases (i.e. $b(\cdot, \cdot)$ is the Fourier basis). Depending on how the particle-particle interactions are treated, a FFT-based summation method belongs to the category

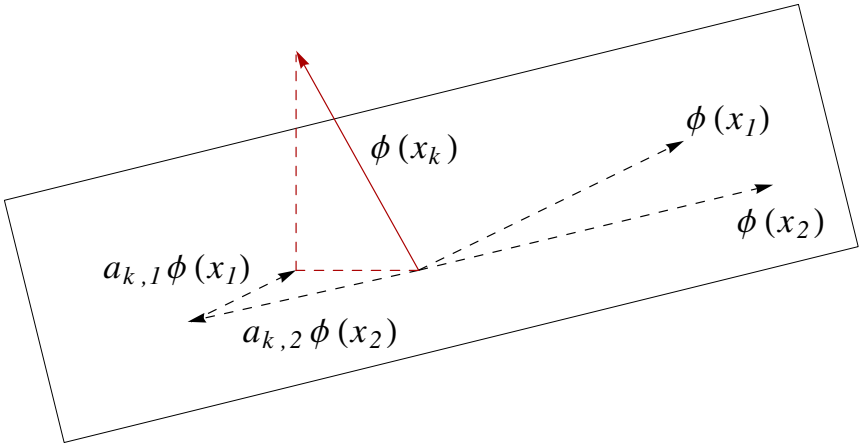


Figure 13: Kernel linear independence criterion for choosing the reduced set expansion. The solid arrow $\phi(\mathbf{x_k})$ is projected down into a set of components $\phi(\mathbf{x_1})$ and $\phi(\mathbf{x_2})$ with some reconstruction error.

of Particle-Particle-Particle Mesh (P^3M) method or Particle-Mesh (PM) method. However, these methods do not scale beyond three dimensions due to uniform grids.

Recently, machine learning practitioners have employed a variant of reduced set method that utilize positive-definiteness (or conditionally positive-definiteness) of the kernel function and successfully scaled many kernel methods such as SVM and GPR. [66] uses the incomplete Cholesky factorization to compute a sparse low rank approximation $\widetilde{\mathbf{K}} = \mathbf{G}\mathbf{G}^T \simeq \mathbf{K}$. However, this requires storing $\mathbf{K}$ in memory. Another simple algorithm is the Nystrom method [195] which reduces the $\mathcal{O}(N^3)$ computational cost by carrying out an eigendecompoisition on a smaller set of randomly chosen training points. Recued set methods generally require optimizing the basis points given a pre-selected error criterion (i.e. on reconstruction error in the reproducing kernel Hilbert space or generalization error with/without regularization) and the resulting dictionary $\mathbf{R}^{reduced}$ can be quite large in some cases [176, 175, 141, 167].

2.3.3 Monte Carlo-based Methods

[95] proposes a probabilistic approximation scheme based on the central limit theorem, and [113] used both deterministic and probabilistic approximations. Especially,

probabilistic approximations can help overcome the *curse of dimensionality* at the expense of indeterminism in approximated kernel sums. The error bounds provided by the deterministic expansions are generally pessimistic and loose. We have an additional parameter α that controls the probability level at which the deviation between each approximation and its corresponding exact values holds.

Theorem 2.3.2. Central limit theorem: *Let f_1, f_2, $\cdots$, f_m be independent, identically distributed samples from the probability distribution F with variance σ^2, and $\widetilde{\mu} = \frac{1}{m}\sum_{s=1}^{m} f_s$ be the sample mean of the samples. As $m \to \infty$, $\widetilde{\mu} \rightsquigarrow N(\mu, \sigma^2/m)$.*

A widely accepted statistical rule of thumb asserts that 30 or more samples are usually enough to put a sample mean into the asymptotic regime. Berry-Esseen theorem characterizes the rate at which this convergence to normality takes place more precisely:

Theorem 2.3.3. Berry-Esseen theorem: *Let $\widetilde{\mu}$ be the sample mean of m samples drawn from the distribution F, and let μ, σ^2, and ρ be the mean, variance, and third central moment of F. Let $F_m(x)$ be the cumulative distribution function of $\widetilde{\mu}$, and $\Psi(x; \mu, \sigma^2)$ be the cumulative distribution function of the Gaussian with mean μ and variance σ^2. Then there exists a constant $C > 0$ such that for all values of $\widetilde{\mu}$ and m:*

$$\left|F_m(\widetilde{\mu}) - \Psi(\widetilde{\mu}; \mu, \sigma^2)\right| \leq \frac{C\rho}{\sigma^3\sqrt{m}}$$

which roughly says that the discrepancy between the normal distribution and the sample mean distribution goes down as $\frac{1}{\sqrt{m}}$. For each $\mathbf{q} \in \mathbf{Q}$, we get the empirical distribution $F_m^{\mathbf{q}}$ using m samples. We can then form an approximate $\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{sub})$:

$$\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{sub}; F_m^{\mathbf{q}}) = |\mathbf{R}^{sub}|\widetilde{\mu}_{F_m^{\mathbf{q}}} = \frac{|\mathbf{R}^{sub}|}{m}\sum_{s=1}^{m} k(\mathbf{q}, \mathbf{r_s}) \tag{2.3.1}$$

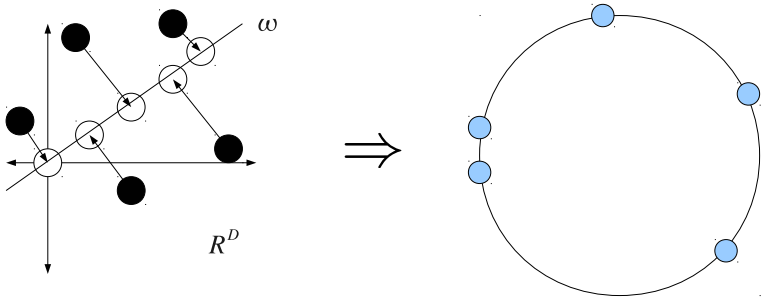


Figure 14: Random feature extraction method by [150].

2.3.4 Random Projection-based Methods

[2] proposes to replace the kernel matrix $\mathbf{K}$ with its randomized variant using random projections and randomized rounding. Randomized rounding sparsifies $\mathbf{K}$, which helps in accelerating the matrix-matrix product inside orthogonal/Lanczos iteration, while the random projection reduces the dimensionality of each point which enables faster computaton of pairwise distances. However, the authors focused only on theoretical analysis without providing concrete experimental results.

[150] proposes explicitly mapping the data to a low-dimensional Euclidean inner product space using a randomized feature map. The inner product in the new low-dimensional space equals the kernel value in expectation. For positive-definite kernels, this techniques relies on the classical theorem by Bochner [158] stated here:

Theorem 2.3.4. *A continuous kernel $k(\mathbf{q}, \mathbf{r}) = k(\mathbf{q} - \mathbf{r})$ on $\mathbb{R}^D$ is positive definite if and only if $k(\delta)$ is the Fourier transform of a non-negative measure.*

A positive continuous real-valued kernel $k(\mathbf{q}, \mathbf{r})$ can now be written as[4]:

$$k(\mathbf{q}, \mathbf{r}) = \int_{\mathbb{R}^D} p(\boldsymbol{\omega}) e^{\imath \cdot \boldsymbol{\omega}^T(\mathbf{q}-\mathbf{r})} d\boldsymbol{\omega} = \int_{\mathbb{R}^D} p(\boldsymbol{\omega}) \cos\left(\boldsymbol{\omega}^T(\mathbf{q}-\mathbf{r})\right) d\omega = \mathbb{E}_{\boldsymbol{\omega}\sim p(\omega)}\left[\boldsymbol{\xi}_{\boldsymbol{\omega}}(\mathbf{q})^T \boldsymbol{\xi}_{\boldsymbol{\omega}}(\mathbf{r})\right] \tag{2.3.2}$$

[4]Although [85] uses the same Fourier integral representation of k, the authors truncate the Fourier integral and apply a quadrature rule for approximation.

where $\boldsymbol{\xi}_{\boldsymbol{\omega}}(\mathbf{x}) = \left[\cos(\boldsymbol{\omega}^T\mathbf{x}), \sin(\boldsymbol{\omega}^T\mathbf{x})\right]^T$. For the Gaussian kernel, $p(\boldsymbol{\omega}) = e^{-||\boldsymbol{\omega}||^2/(2/h)^2}$, the D-dimensional Gaussian function with the bandwidth of $\frac{1}{h}$. We can now express Equation (2.0.5) as:

$$\Phi(\mathbf{q}; \mathbf{R}) = \sum_{\mathbf{r_j} \in \mathbf{R}} w_j \mathbb{E}_{\boldsymbol{\omega} \sim p(\boldsymbol{\omega})} \left[\boldsymbol{\xi}_{\boldsymbol{\omega}}(\mathbf{q})^T \boldsymbol{\xi}_{\boldsymbol{\omega}}(\mathbf{r_j})\right] \tag{2.3.3}$$

This approach has been recently extended to utilize for positive definite dot product kernels of the form $k(\mathbf{x}, \mathbf{y}) = f(< \mathbf{x}, \mathbf{y} >)$ for some real; valued function $f : \mathbb{R} \to \mathbb{R}$ and compositional kernels of the form $k_{co}(\mathbf{x}, \mathbf{y}) = k_{dp}(k(\mathbf{x}, \mathbf{y}))$ where k_{dp} is some dot product kernel and k is an arbitrary positive definite kernel.

For general kernels, [150] proposes the usage of random binning which is reminiscent of the locality sensitive hashing approach [74]. [169] generalizes [150] by providing randomization techniques for structured data including strings and graphs.

2.4 Error Bounds

Many kernel summation algorithms trade precision over speed. The following error bounding criteria on Equation (2.0.3) are used in the literature:

Definition 2.4.1. τ absolute error bound: *For* $\mathbf{x_{i_1}} \in \mathbf{X_1}$, *compute* $\widetilde{\Phi}(\mathbf{x_{i_1}}; \mathbf{X_2} \times \cdots \times \mathbf{X_n})$ *such that* $\left|\widetilde{\Phi}(\mathbf{x_{i_1}}; \mathbf{X_2} \times \cdots \times \mathbf{X_n}) - \Phi(\mathbf{x}; \mathbf{X_2} \times \cdots \times \mathbf{X_n})\right| \leq \tau$.

Definition 2.4.2. ϵ relative error bound: *For* $\mathbf{x_{i_1}} \in \mathbf{X_1}$, *compute* $\widetilde{\Phi}(\mathbf{x_{i_1}}; \mathbf{X_2} \times \cdots \times \mathbf{X_n})$ *such that* $\left|\widetilde{\Phi}(\mathbf{x_{i_1}}; \mathbf{X_2} \times \cdots \times \mathbf{X_n}) - \Phi(\mathbf{x_{i_1}}; \mathbf{X_2} \times \cdots \times \mathbf{X_n})\right| \leq \epsilon \left|\Phi(\mathbf{x_{i_1}}; \mathbf{X_2} \times \cdots \times \mathbf{X_n})\right|$.

Bounding the relative error is much harder because the error bound criterion is in terms of the initially unknown exact quantity. As a result, many previous methods [84, 201] have focused on bounding the absolute error. The relative error bound criterion is preferred to the absolute error bound criterion if the amount of error incurred must be controlled with respect to the relative magnitude of the computed

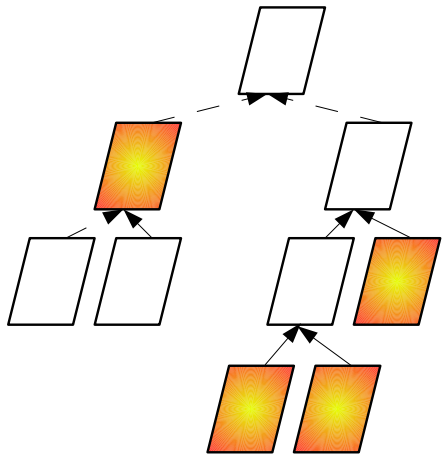


Figure 15: The colored nodes form a frontier of nodes used to approximate $\Phi(\mathbf{q};\mathbf{R}) = \sum_{\mathbf{R}^{sub}} \Phi(\mathbf{q};\mathbf{R}^{sub})$ with $\widetilde{\Phi}(\mathbf{q};\mathbf{R}) = \sum_{\mathbf{R}^{sub}} \widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub})$.

quantities. The following error bound criterion is a hybrid form that enforces both relative and absolute errors.

Definition 2.4.3. $(1-\alpha)$ **probabilistic** ϵ **relative/**τ **absolute error:** *For* $\mathbf{x_{i_1}} \in \mathbf{X_1}$, *compute* $\widetilde{\Phi}(\mathbf{x_{i_1}};\mathbf{X_2}\times\cdots\times\mathbf{X_n})$, *such that with at least probability* $0 < 1-\alpha \leq 1$, $\left|\widetilde{\Phi}(\mathbf{x_{i_1}};\mathbf{X_2}\times\cdots\times\mathbf{X_n}) - \Phi(\mathbf{x_{i_1}};\mathbf{X_2}\times\cdots\times\mathbf{X_n})\right| \leq \epsilon\left|\Phi(\mathbf{x_{i_1}};\mathbf{X_2}\times\cdots\times\mathbf{X_n})\right| + \tau$.

Using Trees for Allocating Errors. Suppose we are approximating Equation (2.0.5) and given a tree over the reference set R (see Figure 15). We can use a frontier of nodes in this tree to form a partition of the entire reference set $\mathbf{R}$ such that $\mathbf{R} = \bigcup \mathbf{R}^{sub}$. Given that the exact quantity for a given query $\mathbf{q}$ is given by the sum of the exact quantities contributed by the partition: $\Phi(\mathbf{q};\mathbf{R}) = \sum_{\mathbf{R}^{sub}} \Phi(\mathbf{q};\mathbf{R}^{sub})$, we can approximate the contribution of each part and aggregate them to form the overall approximation: $\widetilde{\Phi}(\mathbf{q};\mathbf{R}) = \sum_{\mathbf{R}^{sub}} \widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub})$. The maximum error is then by bounded by the application of triangle inequality:

$$\left|\widetilde{\Phi}(\mathbf{q};\mathbf{R}) - \Phi(\mathbf{q};\mathbf{R})\right| = \left|\sum_{\mathbf{R}^{sub}} \widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub}) - \Phi(\mathbf{q};\mathbf{R}^{sub})\right| \leq \sum_{\mathbf{R}^{sub}} \left|\widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub}) - \Phi(\mathbf{q};\mathbf{R}^{sub})\right| \tag{2.4.1}$$

Each part $\left|\widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub}) - \Phi(\mathbf{q};\mathbf{R}^{sub})\right|$ can be bounded using any of the approximation methods in Section 2.3 given the selected overall error bound criterion (see Section 2.4). The usual strategy is to allocate errors in proportion to 1) the number of points in the given frontier node; 2) the amount of data variance in the given frontier node [95]. The error allocation can be dynamically re-adjusted during the computation [112].

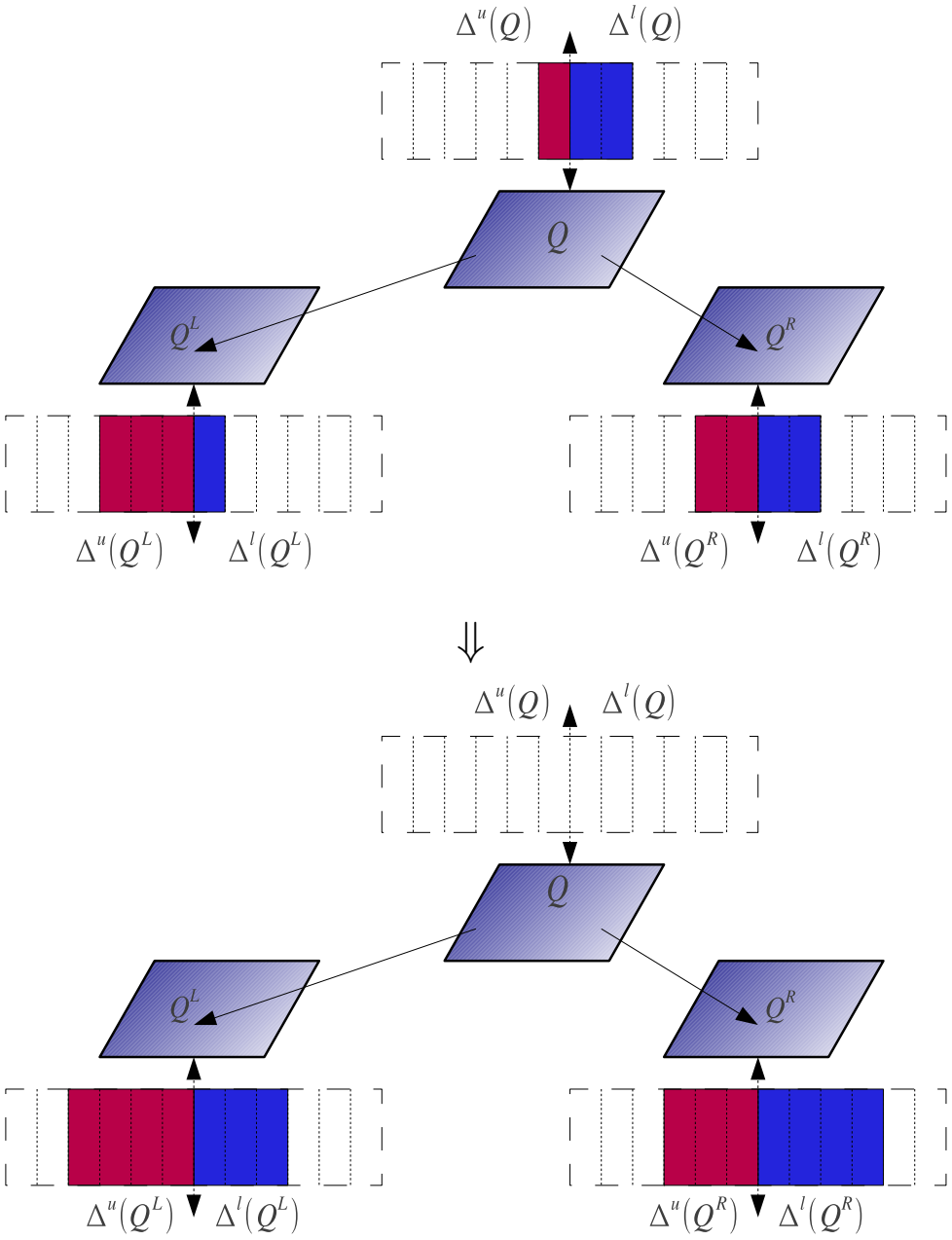


Figure 16: Bound propagation example. An internal node pushes quantities to its immediate child nodes. These child nodes in turn incorporate the received quantities inside their appropriate slots.

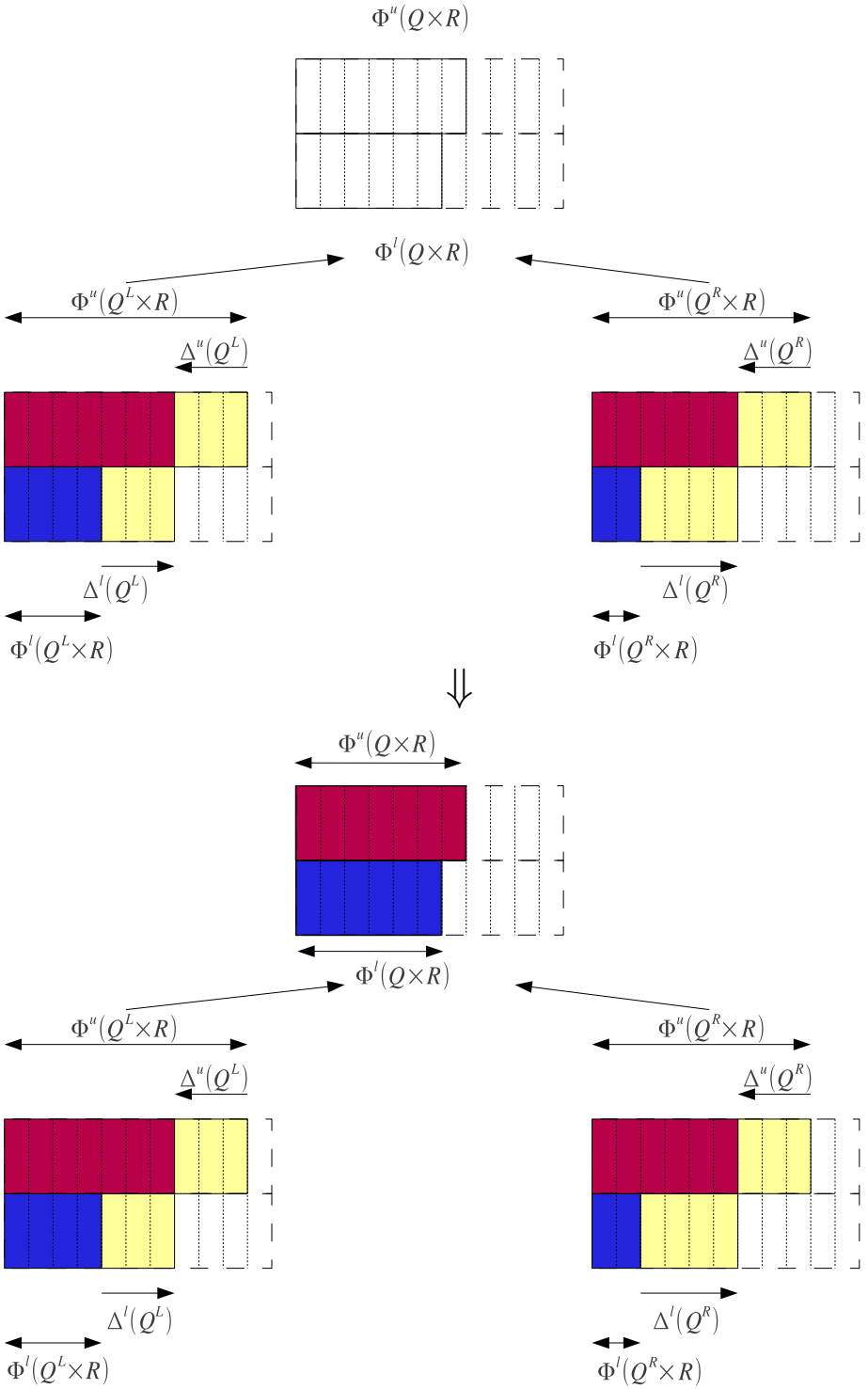


Figure 17: Bound refinement example. An internal node refines its lower and upper bounds based on the bounds held by its child nodes.

CHAPTER III

SERIES EXPANSION-BASED METHOD I

This chapter introduces the series expansion-based method for scaling pairwise kernel summations (Equation (2.0.5)). We focus on a specific instance of Equation (2.0.5) where the kernel is the Gaussian kernel $k(\mathbf{q}, \mathbf{r}) = \exp\left(\frac{-||\mathbf{q}-\mathbf{r}||^2}{2h^2}\right)$:

$$\mathbf{\Phi}\left(\mathbf{Q} \times \mathbf{R}\right) = \underset{\mathbf{q_i} \in \mathbf{Q}}{\mathbf{map}} \left(\sum_{\mathbf{r_j} \in \mathbf{R}} w_j \exp\left(\frac{-||\mathbf{q_i} - \mathbf{r_j}||^2}{2h^2}\right)\right) \tag{3.0.2}$$

where we put a uniform weight over each reference point (i.e. $w_j = 1$)[1]. Intuitively, selecting the Gaussian kernel already puts some structures to the problem. The Gaussian kernel is 1) continuously differentiable with respect to its arguments; 2) positive-definite and has a well-defined Fourier transform. Fast multipole methods discussed in this chapter take advantage of the first property. Here we define the computational task tackled in this chapter.

Problem: Suppose we are given the set of query points $\mathbf{Q}$ and the set of reference points $\mathbf{R}$. Given a pairwise Gaussian kernel function $k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-||\mathbf{x}-\mathbf{y}||^2}{2h^2}\right)$, the relative error level $\epsilon > 0$, and the desired kernel sum $\Phi(\mathbf{q}; \mathbf{R}) = \sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{q}, \mathbf{r_j})$ for each $\mathbf{q} \in \mathbf{Q}$,

Task: Compute an approximation $\widetilde{\Phi}(\mathbf{q}; \mathbf{R})$ for each $\mathbf{q} \in \mathbf{Q}$ such that $\left|\widetilde{\Phi}(\mathbf{q}; \mathbf{R}) - \Phi(\mathbf{q}; \mathbf{R})\right| \leq \epsilon \Phi(\mathbf{q}; \mathbf{R})$ as fast as possible.

[1] The extension to non-uniform weights is trivial.

This chapter builds on [114] where the Dual-Tree Fast Gauss Transform was presented briefly. The Dual-tree Fast Gauss Transform was the first serious effort in combining the best tools from discrete algorithms and continuous approximation theory. This chapter adds details on the approximation mechanisms used in the algorithm and provides a more thorough comparison with the other algorithms. Section 3.1 describes the extensions to the dual-tree algorithm to handle higher-order series expansion approximations. This expansion uses the $\mathcal{O}(p^D)$ expansion proposed in [84] and results from taking dimension-wise products of truncated Taylor expansions. In Section 3.2, we provide performance comparison with some of the existing methods for evaluating the Gaussian kernel sums in a serial setting.

3.1 Dual-Tree Fast Gauss Transform

3.1.1 Mathematical Preliminaries

Univariate Taylor's Theorem. The univariate Taylor's theorem is crucial for the approximation mechanism in Fast Gauss transform and the new algorithm:

Theorem 3.1.1. *If $n \geq 0$ is an integer and f is a function which is n times continuously differentiable on the closed interval $[c, x]$ and $n+1$ times differentiable on (c, x) then*

$$f(x) = \sum_{i=0}^{n} f^{(i)}(c) \frac{(x-c)^i}{i!} + R_n \tag{3.1.1}$$

where the Lagrange form of the remainder term is given by
$R_n = f^{(n+1)}(\xi) \frac{(x-c)^{n+1}}{(n+1)!}$ for some $\xi \in (c, x)$.

Properties of the Gaussian Kernel. Based on the univariate Taylor's Theorem stated above, [84] develops the series expansion mechanism for the Gaussian kernel sum. Our development begins with one-dimensional setting and generalizes to multi-dimensional setting. The Rodrigues' formula defines the Hermite polynomials:

$$H_n(t) = (-1)^n \exp(t^2) D^n \exp(-t^2), t \in \mathbb{R}^1 \tag{3.1.2}$$

The first few polynomials include: $H_0(t) = 1,\ H_1(t) = 2t,\ H_2(t) = 4t^2 - 2$. The generating function for Hermite polynomials is defined by:

$$\exp(2ts - s^2) = \sum_{n=0}^{\infty} \frac{s^n}{n!} H_n(t) \tag{3.1.3}$$

Let us define the Hermite functions $h_n(t) = \exp(-t^2)H_n(t)$. Multiplying both sides of Equation (3.1.3) by $\exp(-t^2)$ yields:

$$\exp(-(t-s)^2) = \sum_{n=0}^{\infty} \frac{s^n}{n!} h_n(t) \tag{3.1.4}$$

We would like to use a "scaled and shifted" version of this derivation for taking the bandwidth h into account.

$$\exp\left(\frac{-(t-s)^2}{2h^2}\right) = \exp\left(\frac{-((t-s_0)-(s-s_0))^2}{2h^2}\right) = \sum_{n=0}^{\infty} \frac{1}{n!} \left(\frac{s-s_0}{\sqrt{2h^2}}\right)^n h_n\left(\frac{t-s_0}{\sqrt{2h^2}}\right)$$

Note that our D-dimensional multivariate Gaussian kernel can be expressed as a product of D one-dimensional Gaussian kernel. Similarly, the multidimensional Hermite functions can be written as a product of one-dimensional Hermite functions using the following identity for any $\mathbf{t} \in \mathbb{R}^D$.

$$\begin{aligned} H_{\boldsymbol{\alpha}}(\mathbf{t}) &= H_{\boldsymbol{\alpha}[1]}(\mathbf{t}[1]) \cdots H_{\boldsymbol{\alpha}[D]}(\mathbf{t}[D]) \\ h_{\boldsymbol{\alpha}}(\mathbf{t}) &= \exp(-||\mathbf{t}||^2) H_{\boldsymbol{\alpha}}(\mathbf{t}) = h_{\boldsymbol{\alpha}[1]}(\mathbf{t}[1]) \cdots h_{\boldsymbol{\alpha}[D]}(t[D]) \end{aligned} \tag{3.1.5}$$

$$\exp\left(\frac{-||\mathbf{t}-\mathbf{s}||^2}{2h^2}\right) = \prod_{d=1}^{D} \exp\left(\frac{-(\mathbf{t}[d]-\mathbf{s}[d])^2}{2h^2}\right) \tag{3.1.6}$$

We can also express the multivariate Gaussian about another point $\mathbf{s_0} \in \mathbb{R}^D$ as:

$$\begin{aligned} \exp\left(\frac{-||\mathbf{t}-\mathbf{s}||^2}{2h^2}\right) &= \prod_{d=1}^{D} \left(\sum_{n_d=0}^{\infty} \frac{1}{n_d!} \left(\frac{\mathbf{s}[d]-\mathbf{s_0}[d]}{\sqrt{2h^2}}\right)^{n_d} h_{n_d}\left(\frac{\mathbf{t}[d]-\mathbf{s_0}[d]}{\sqrt{2h^2}}\right) \right) \\ &= \sum_{\boldsymbol{\alpha} \geq 0} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{s}-\mathbf{s_0}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}} h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{t}-\mathbf{s_0}}{\sqrt{2h^2}}\right) \end{aligned} \tag{3.1.7}$$

The representation which is *dual* to Equation (3.1.7) is given by:

$$\begin{aligned} \exp(\frac{-||\mathbf{t}-\mathbf{s}||^2}{2h^2}) &= \prod_{d=1}^{D} \left(\sum_{n_d=0}^{\infty} \frac{(-1)^{n_d}}{n_d!} h_{n_d}\left(\frac{\mathbf{t_0}[d]-\mathbf{s}[d]}{\sqrt{2h^2}}\right) \left(\frac{\mathbf{t}[d]-\mathbf{t_0}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \right) \\ &= \sum_{\boldsymbol{\beta} \geq 0} \frac{(-1)^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}}\left(\frac{\mathbf{t_0}-\mathbf{s}}{\sqrt{2h^2}}\right) \left(\frac{\mathbf{t}-\mathbf{t_0}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \end{aligned} \tag{3.1.8}$$

The final property is the recurrence relation of the one-dimensional Hermite function:

$$h_{n+1}(t) = 2t \cdot h_n(t) - 2n \cdot h_{n-1}(t), t \in \mathbb{R}^1 \tag{3.1.9}$$

and the Taylor expansion of the Hermite function $h_{\boldsymbol{\alpha}}(\mathbf{t})$ about $\mathbf{t_0} \in \mathbb{R}^D$.

$$h_{\boldsymbol{\alpha}}(\mathbf{t}) = \sum_{\boldsymbol{\beta} \geq 0} \frac{(\mathbf{t} - \mathbf{t_0})^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!}(-1)^{|\boldsymbol{\beta}|} h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}(\mathbf{t_0}) \tag{3.1.10}$$

3.1.2 Notations in Algorithm Descriptions

Here we summarize notations used throughout the descriptions and the pseudocodes for our algorithms. The followings are notations that are relevant to a query point $\mathbf{q_i} \in \mathbf{Q}$ or a query node $\mathbf{Q}^{sub}$ in the query tree.

- $\mathbf{R}^{\mathsf{E}}(\cdot)$: The set of reference points $\mathbf{r_{j_n}} \in \mathbf{R}$ whose pairwise interaction is computed exhaustively for a query point $\mathbf{q_i} \in \mathbf{Q}$ or a query node $\mathbf{Q}^{sub}$.

- $\mathbf{R}^{\mathsf{F2L}}(\cdot)$: The set of reference points $\mathbf{r_{j_n}} \in \mathbf{R}$ whose contribution is via far-to-local approximation for a given query point $\mathbf{q_i} \in \mathbf{Q}^{sub}$ or a query node $\mathbf{Q}^{sub}$.

- $\mathbf{R}^{\mathsf{DL}}(\cdot)$: The set of reference points $\mathbf{r_{j_n}} \in \mathbf{R}$ whose contribution is from a local expansion for a given query point $\mathbf{q_i} \in \mathbf{Q}^{sub}$ or a query node $\mathbf{Q}^{sub}$.

- $\mathbf{R}^{\mathsf{DF}}(\cdot)$: The set of reference points $\mathbf{r_{j_n}} \in \mathbf{R}$ whose contribution is from a far-field expansion for a given query point $\mathbf{q_i} \in \mathbf{Q}^{sub}$ or a query node $\mathbf{Q}^{sub}$.

The followings are notations relevant to a query point $\mathbf{q_i} \in \mathbf{Q}$.

- $\Phi(\mathbf{q_i}; \mathbf{R}^{sub})$: The true initially unknown kernel sum for a query point $\mathbf{q_i}$ contributed by the reference set $\mathbf{R}^{sub} \subseteq \mathbf{R}$, i.e. $\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} k(||\mathbf{q_i} - \mathbf{r_{j_n}}||)$.

- $\Phi^l(\mathbf{q_i}; \mathbf{R})$: A lower bound on $\Phi(\mathbf{q_i}; \mathbf{R})$.

- $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})$: A lower bound on $\Phi(\mathbf{q_i}; \mathbf{R})$ for $\mathbf{q_i} \in \mathbf{Q}^{sub}$.

- $\Phi^u(\mathbf{q_i}; \mathbf{R})$: An upper bound on $\Phi(\mathbf{q_i}; \mathbf{R})$.

- $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R})$: An upper bound on $\Phi(\mathbf{q_i}; \mathbf{R})$ for $\mathbf{q_i} \in \mathbf{Q}^{sub}$.

- $\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R}^{sub})$: An approximation to $\Phi(\mathbf{q_i}; \mathbf{R}^{sub})$ for $\mathbf{R}^{sub} \subseteq \mathbf{R}$. The additive property for a family of pairwise disjoint sets $\{\mathbf{R_i}\}_{i=1}^m$: $\widetilde{\Phi}\left(\mathbf{q_i}; \bigcup_{i=1}^{m} \mathbf{R_i}\right) = \sum_{i=1}^{m} \widetilde{\Phi}(\mathbf{q_i}; \mathbf{R_i})$.

- $\widetilde{\Phi}\left(\mathbf{q_i}; \{(\mathbf{R}^j, \mathsf{A_j})\}_{j=1}^m\right)$: A refined notation of $\widetilde{\Phi}\left(\mathbf{q_i}; \bigcup_{j=1}^{m} \mathbf{R}^j\right)$ to specify the type of approximation $\mathsf{A_j}$ used for each reference node $\mathbf{R}^j$.

Here we define some notations for representing postponed bound changes to $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$ and $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$ for all $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub} \subseteq \mathbf{Q}$.

- $\Delta^l(\mathbf{Q}^{sub})$: Postponed lower bound changes on $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})$ for a query node $\mathbf{Q}^{sub}$ in the query tree and $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$ for $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$.

- $\Delta^u(\mathbf{Q}^{sub})$: Postponed upper bound changes on $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R})$ for a query node $\mathbf{Q}^{sub}$ in the query tree and $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$ for $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$.

These postponed changes to the upper and lower bounds must be incorporated into each individual query $\mathbf{q_{i_m}}$ belonging to the sub-tree under $\mathbf{Q}^{sub}$.

Our series-expansion based algorithm uses four different approximation methods, i.e. $\mathsf{A} \in \{\mathsf{D}, \widetilde{\mathsf{N}}(\mathbf{c}, p), \mathsf{F}(\mathbf{c}, p), \mathsf{N}(\mathbf{c}, p)\}$. For each $\mathbf{R}^{sub}$, an approximation method is chosen. D denotes the exhaustive computation of $\sum_{\mathbf{r_{j_n}} \in \mathbf{R}} k(||\mathbf{q_i} - \mathbf{r_{j_n}}||)$. $\widetilde{\mathsf{N}}(\mathbf{c}, p)$ denotes the translation of the order p far-field moments of $\mathbf{R}^{sub}$ to the local moments in the query node $\mathbf{Q}^{sub}$ that owns $\mathbf{q_i}$ about a representative centroid $\mathbf{c}$ inside $\mathbf{Q}^{sub}$. $\mathsf{F}(\mathbf{c}, p)$ denotes the evaluation up to the p-th order far-field expansion formed by the moments of $\mathbf{R}^{sub}$ expanded about a representative point $\mathbf{c}$ inside $\mathbf{R}^{sub}$. $\mathsf{N}(\mathbf{c}, p)$ denotes the p-th order direct accumulation of the local moments due to $\mathbf{R}^{sub}$ about a representative centroid $\mathbf{c}$ inside $\mathbf{Q}^{sub}$ that owns $\mathbf{q_i}$. We discuss these methods in Section 3.1.3.

3.1.3 Series Expansion for the Gaussian Kernel Sums

We would like to point out to our readers that we present the series expansion in a way that sheds light to a working implementation. [84] chose a theorem-proof format for explaining the essential operations. We present the series expansion methods from the more informed computer science perspective of divide-and-conquer and data structures, where the discrete aspects of the methods are concerned.

One can derive the series expansion for the Gaussian kernel sums (defined in Equation (3.0.2)) using Equation (3.1.7) and Equation (3.1.8). The basic idea is to express the kernel sum contribution of a reference node as a Taylor series of infinite terms and truncate it after some number of terms, given that the truncation error meets the desired absolute error tolerance.

The followings are two main types of Taylor series representations for an infinitely differentiable kernel $k(\cdot)$. The key difference between two representations is the location of the expansion center which is either in a reference region or a query region. The expansion center for both expansion types is conveniently chosen to be the geometric center of the region. These representations were briefly introduced in Section 2.3.1.

Far-field Expansion. This is derived from Equation (3.1.7) and expresses the kernel sum contribution from the reference points in $\mathbf{R}^{sub}$ for an arbitrary query point. It is expanded about $\mathbf{c_R}$, a representative point of $\mathbf{R}^{sub}$. Equation (3.1.7) is an infinite series, and thus we impose a truncation order p along each dimension. Substituting

$\mathbf{q_i}$ for $\mathbf{t}$, $\mathbf{r_j}$ for $\mathbf{s}$ and $\mathbf{c_R}$ for $\mathbf{s_0}$ into Equation (3.1.7) yields:

$$\begin{aligned}
\Phi(\mathbf{q_i}; \mathbf{R}^{sub}) &= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \exp\left(\frac{-||\mathbf{q_i} - \mathbf{r_{j_n}}||^2}{2h^2}\right) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \prod_{d=1}^{D} \left(\sum_{\boldsymbol{\alpha}[d]=0}^{\infty} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right) \right) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \prod_{d=1}^{D} \left(\sum_{\boldsymbol{\alpha}[d]<p} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d)]}{\sqrt{2h^2}}\right) + \right. \\
&\qquad \left. \sum_{\boldsymbol{\alpha}[d]\geq p} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right) \right)
\end{aligned}$$

Truncating after p terms along each dimension yields:

$$\begin{aligned}
&\Phi(\mathbf{q_i}; \mathbf{R}^{sub}) \approx \widetilde{\Phi}(\mathbf{q_i}, \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \prod_{d=1}^{D} \left(\sum_{\boldsymbol{\alpha}[d]\leq p} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{r_{j_n}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right) \right) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \sum_{\boldsymbol{\alpha}\leq p} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}} h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right) \\
&= \sum_{\boldsymbol{\alpha}\leq p} \left[\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}} \right] h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right) \\
&= \sum_{\boldsymbol{\alpha}\leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)
\end{aligned}$$

where we denote

$$M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}} \tag{3.1.11}$$

which is a function of a reference node $\mathbf{R}^{sub}$ and an expansion center $\mathbf{c}$. We denote $\widetilde{\Phi}(\mathbf{q_i}; \{\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}, p)\})$ as the **far-field expansion of order p for the kernel sum contribution of $\mathbf{R}^{sub}$ expanded about $\mathbf{c}$**. Ideally, we would like to choose the smallest p such that the truncation after the chosen order p incurs tolerable error; this will be discussed in Section 3.1.5. Note that the far-field expansion for the Gaussian kernel separates the interaction between a reference point and a query point (namely

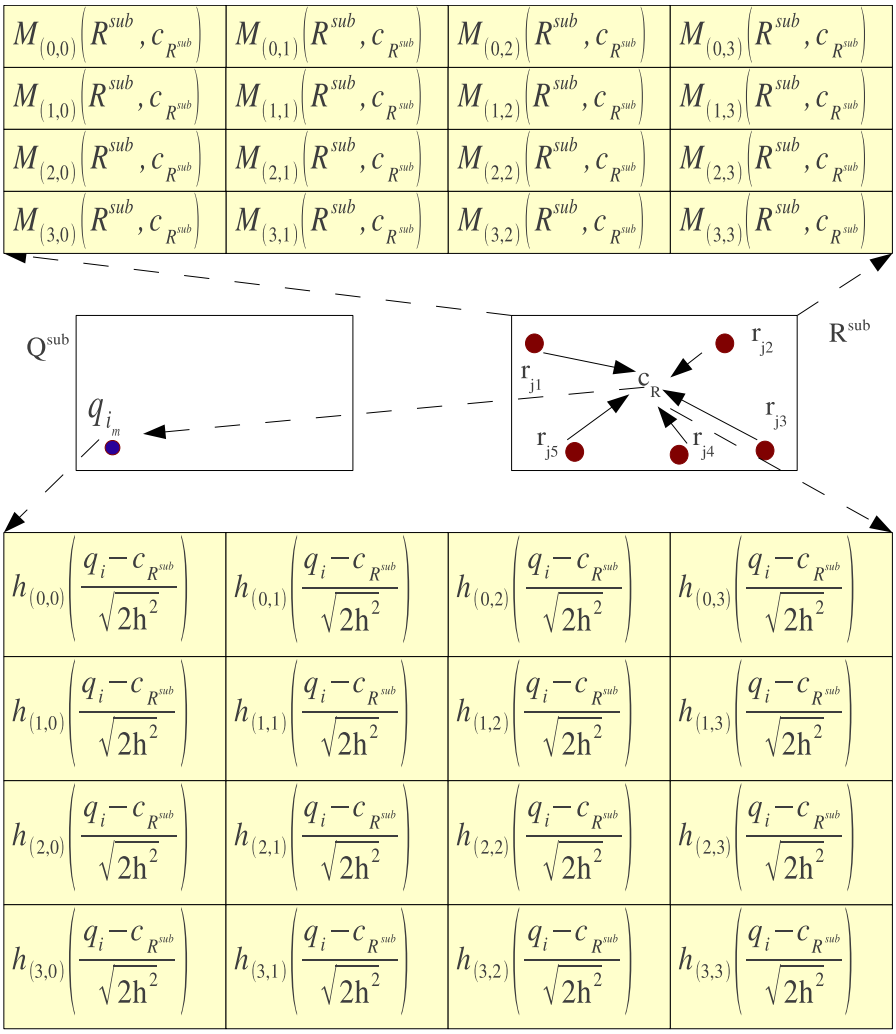


Figure 18: Given the query node $\mathbf{Q}^{sub}$ containing the query points $\{\mathbf{q_{i_m}}\}_{m=1}^{|\mathbf{Q}^{sub}|}$ and the reference node $\mathbf{R}^{sub}$ containing the reference points $\{\mathbf{r_{j_n}}\}_{n=1}^{|\mathbf{R}^{sub}|}$, evaluating the far-field expansion generated by the reference points at the given query point $\mathbf{q_{i_m}}$ up to four terms in each dimension, $\Phi(\mathbf{q_{i_m}}; \mathbf{R}^{sub}) \approx \widetilde{\Phi}(\mathbf{q_{i_m}}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, 3))\}) = \sum_{\boldsymbol{\alpha} \leq 3} \left[\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}} \right] h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q_{i_m}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$, involves computing the sum of the element-wise product between the two-dimensional array of far-field coefficients with the query-dependent two-dimensional array.

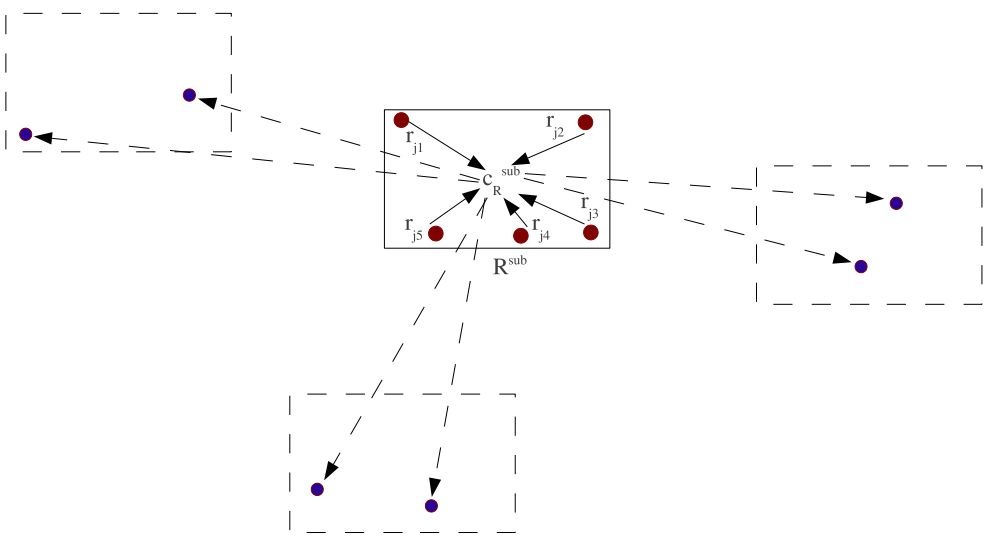


Figure 19: The Gaussian kernel sum series expansion represented by the far-field coefficients in $\mathbf{R}^{sub}$, $\sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$, is valid regardless of the location of the given query point, given the size constraint on the reference node (see Section 3.1.5). Each query point location will incur different amount of error.

$\exp\left(-||\mathbf{q_i} - \mathbf{r_{j_n}}||^2/(2h^2)\right)$ into a summation of two product terms. For each multi-index $\boldsymbol{\alpha}$, $M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}})$, which depends only on the *intrinsic* information for the reference node (the reference points $\mathbf{r_{j_n}} \in \mathbf{R}^{sub}$ and the reference centroid $\mathbf{c}_{\mathbf{R}^{sub}}$ which is constant with respect to $\mathbf{R}^{sub}$), is called the *far-field moments/coefficients* of the reference region $\mathbf{R}^{sub}$. Because $M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}})$ part of the far-field expansion of the Gaussian kernel sums is the same regardless of the query point $\mathbf{q_i}$ used for evaluation, they can be computed **only once** and stored within $\mathbf{R}^{sub}$ for efficiently approximating the contribution of $\mathbf{R}^{sub}$ for different query points (see Figure 18). Precomputing the far-field moments for a reference node $\mathbf{R}^{sub}$ up to $\mathcal{O}(p^D)$ terms (i.e. computing $\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}}$ for each $\boldsymbol{\alpha} \leq p$) requires $\mathcal{O}(|\mathbf{R}^{sub}|p^D)$ operations.

The far-field expansion of order p for the Gaussian kernel sums is valid for any query locations $\mathbf{q_i}$ given that the reference node meets the certain size constraint (see Section 3.1.5). However, for a fixed order p, evaluating on query points that are far away from the reference centroid in general incur smaller amount of error.

Local Expansion. : A *local expansion* (derived from Equation (3.1.8)) is a Taylor expansion of the kernel sums about a representative point $\mathbf{c}_{\mathbf{Q}^{sub}}$ in a query region $\mathbf{Q}$. After substituting $\mathbf{q_i}$ for $\mathbf{t}$, $\mathbf{c}_{\mathbf{Q}^{sub}}$ for $\mathbf{t_0}$ and $\mathbf{r_{j_n}}$ for $\mathbf{s}$, the kernel sum contribution of $\mathbf{R}^{sub}$ to a query point $\mathbf{q_i} \in \mathbf{Q}^{sub}$ is given by:

$$\begin{aligned}
&\Phi(\mathbf{q_i}; \mathbf{R}^{sub}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \exp\left(\frac{-||\mathbf{q_i} - \mathbf{r_{j_n}}||^2}{2h^2}\right) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \prod_{d=1}^{D} \left(\sum_{n_d=0}^{\infty} \frac{(-1)^{n_d}}{n_d!} h_{n_d} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d] - \mathbf{r_{j_n}}[d]}{\sqrt{2h^2}}\right) \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \right) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \prod_{d=1}^{D} \left(\sum_{n_d \le p} \frac{(-1)^{n_d}}{n_d!} h_{n_d} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d] - \mathbf{r_{j_n}}[d]}{\sqrt{2h^2}}\right) \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} + \right. \\
&\qquad \left. \sum_{n_d > p} \frac{(-1)^{n_d}}{n_d!} h_{n_d} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d] - \mathbf{r_{j_n}}[d]}{\sqrt{2h^2}}\right) \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \right)
\end{aligned}$$

Again, truncating after p terms along each dimension yields:

$$\begin{aligned}
&\widetilde{\Phi}(\mathbf{q_i}; \{(\mathbf{R}^{sub}, \mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \prod_{d=1}^{D} \left(\sum_{n_d \le p} \frac{(-1)^{n_d}}{n_d!} h_{n_d} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d] - \mathbf{r_{j_n}}[d]}{\sqrt{2h^2}}\right) \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \right) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \sum_{\boldsymbol{\beta} \le p} \frac{(-1)^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{r_{j_n}}}{\sqrt{2h^2}}\right) \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \\
&= \sum_{\boldsymbol{\beta} \le p} \left[\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{(-1)^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{r_{j_n}}}{\sqrt{2h^2}}\right) \right] \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \\
&= \sum_{\boldsymbol{\beta} < p} N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}
\end{aligned}$$

where we denote:

$$N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) = \begin{cases} \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{(-1)^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}} \left(\frac{\mathbf{c} - \mathbf{r_{j_n}}}{\sqrt{2h^2}}\right) & , \boldsymbol{\beta} \le p \\ 0 & , \text{otherwise} \end{cases} \tag{3.1.12}$$

$\{N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta}}$ are the *direct local moments* of $\mathbf{R}^{sub}$ accumulated at $\mathbf{c}_{\mathbf{Q}^{sub}}$

up to order p. The error bound criterion will be discussed in Section 3.1.5. Note that:

$$\widetilde{\Phi}\left(\mathbf{q_i}; \bigcup_a \left\{(\mathbf{R}^a, \mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}}, p_a))\right\}\right) = \sum_a \widetilde{\Phi}(\mathbf{q_i}, \{(\mathbf{R}^a, \mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}}, p_a))\})$$

$$= \sum_a \sum_{\boldsymbol{\beta} \le p_a} \left[\sum_{\mathbf{r_{jn}} \in \mathbf{R}^a} \frac{(-1)^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{r_{jn}}}{\sqrt{2h^2}} \right) \right] \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$$

$$= \sum_{\boldsymbol{\beta} \le \max_a p_a} \left[\sum_a N_{\boldsymbol{\beta}}(\{(\mathbf{R}^a, (\mathbf{c}_{\mathbf{Q}^{sub}}, p_a))\}) \right] \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$$

$$= \sum_{\boldsymbol{\beta} \le \max_a p_a} N_{\boldsymbol{\beta}} \left(\bigcup_a \{(\mathbf{R}^a, (\mathbf{c}_{\mathbf{Q}^{sub}}, p_a))\} \right) \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$$

In other words, the local moments for a fixed query node $\mathbf{Q}^{sub}$ are additive (see Figure 21) across a set of disjoint portions of the reference dataset $\mathbf{R}$ since its basis functions $\left\{ \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}} \right\}_{\boldsymbol{\beta}}$ remain the same for all reference points regardless of their locations. For a given reference node $\mathbf{R}^{sub}$, accumulating the local moments of $\mathbf{R}^{sub}$ up to p^D terms (that is, evaluating for each $\boldsymbol{\beta} \le p$) requires $\mathcal{O}(|\mathbf{R}^{sub}|p^D)$ operations. These local coefficients are accumulated and stored within the given query node. The induced local expansion is valid for all query points within the query node under certain constraints.

3.1.4 Gaussian Sum Approximation Using Series Expansion

Now again assume we are given a query node $\mathbf{Q}^{sub}$ and a reference node $\mathbf{R}^{sub}$. Here we describe three main methods that use the two expansion types for approximating Gaussian summation, $\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{sub})$, for each $\mathbf{q} \in \mathbf{Q}^{sub}$.

Evaluating a far-field expansion of $\mathbf{R}^{sub}$: Given the pre-computed far-field moments $M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}})$ up to $\mathcal{O}(p^D)$ terms, one could evaluate the far-field expansion for a given query point $\mathbf{q}$ (that is, approximate $\widetilde{\Phi}(\mathbf{q}, \mathbf{R}^{sub})$) by forming a dot-product between the query-dependent vector and the far-field moments, as shown in Figure 18 and Figure 19. Approximating $\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{sub})$ for all $\mathbf{q} \in \mathbf{Q}^{sub}$ requires $\mathcal{O}(|\mathbf{Q}^{sub}|p^D)$ operations.

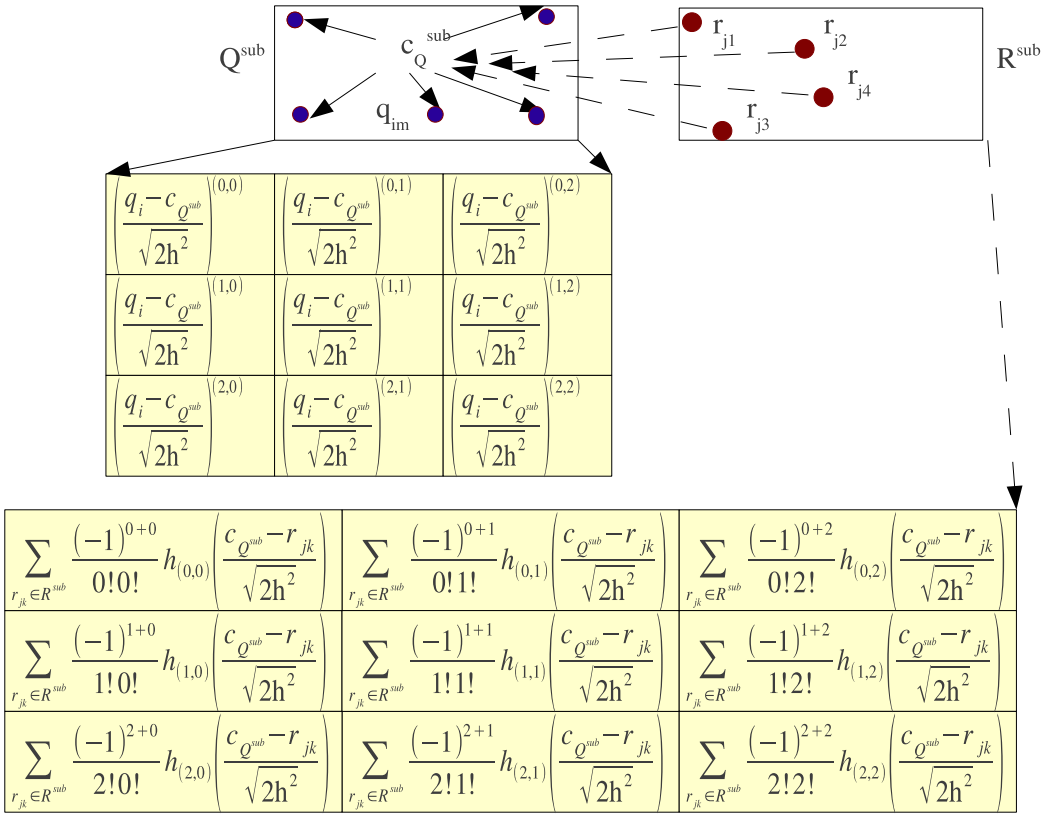


Figure 20: Given the query node $\mathbf{Q}^{sub}$ containing the query points $\{\mathbf{q_{i_m}}\}_{m=1}^{|\mathbf{Q}^{sub}|}$ and the reference node $\mathbf{R}^{sub}$ containing the reference points $\{\mathbf{r_{j_n}}\}_{n=1}^{|\mathbf{R}^{sub}|}$, evaluating the local expansion generated by the reference points at the given query point $\mathbf{q_{i_m}}$ up to third terms in each dimension, $\Phi(\mathbf{q_{i_m}}; \mathbf{R}^{sub}) \approx \widetilde{\Phi}(\mathbf{q_{i_m}}; \{(\mathbf{R}^{sub}, \mathsf{N}(\mathbf{c_{Q^{sub}}}, 2))\}) = \sum_{\boldsymbol{\beta} \leq 2} \left[\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{(-1)^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}} \left(\frac{\mathbf{c_{Q^{sub}}} - \mathbf{r_{j_n}}}{\sqrt{2h^2}} \right) \right] \left(\frac{\mathbf{q_{i_m}} - \mathbf{c_{Q^{sub}}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$, involves taking the dot-product between the two-dimensional array of local coefficients with the query-dependent two-dimensional array.

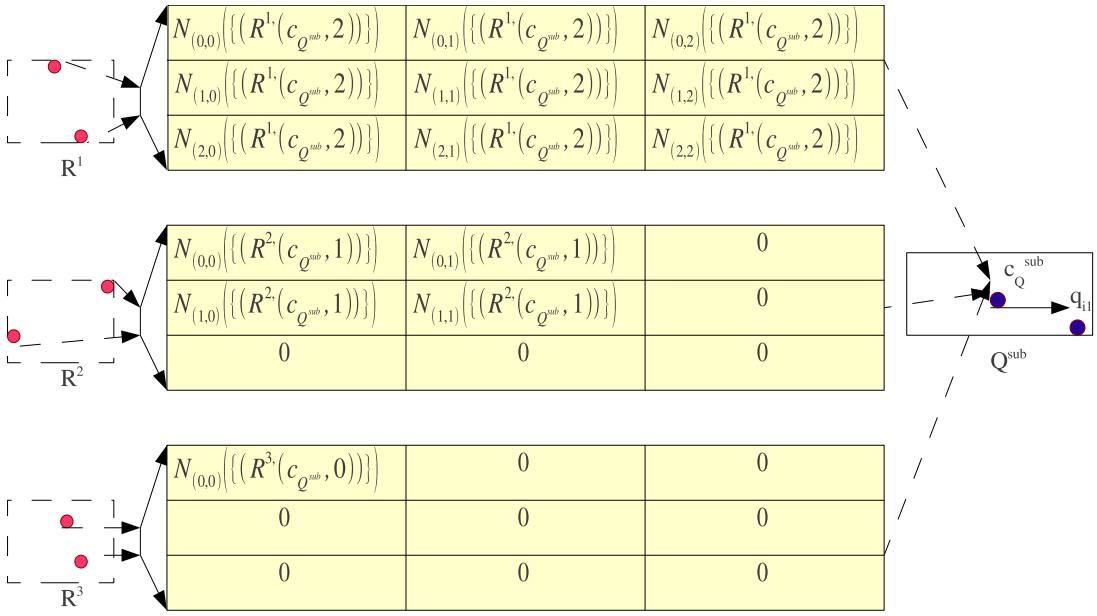


Figure 21: Accumulating direct local moments from three reference nodes $\mathbf{R}^1$, $\mathbf{R}^2$, and $\mathbf{R}^3$ contributing nine terms, four terms, and one term respectively to form the local moments containing the contribution from $\mathbf{R}^1$, $\mathbf{R}^2$, and $\mathbf{R}^3$: $N(\{(\mathbf{R}^1, (\mathbf{c}_{\mathbf{Q}^{sub}}, 2)), (\mathbf{R}^2, (\mathbf{c}_{\mathbf{Q}^{sub}}, 1)), (\mathbf{R}^3, (\mathbf{c}_{\mathbf{Q}^{sub}}, 0))\})$. Zeros denote the positions that are not explicitly computed using Equation (3.1.12). $N(\{(\mathbf{R}^1, (\mathbf{c}_{\mathbf{Q}^{sub}}, 2)), (\mathbf{R}^2, (\mathbf{c}_{\mathbf{Q}^{sub}}, 1)), (\mathbf{R}^3, (\mathbf{c}_{\mathbf{Q}^{sub}}, 0))\}) = N(\{(\mathbf{R}^1, (\mathbf{c}_{\mathbf{Q}^{sub}}, 2))\}) + N(\{(\mathbf{R}^2, (\mathbf{c}_{\mathbf{Q}^{sub}}, 1))\}) + N(\{(\mathbf{R}^3, (\mathbf{c}_{\mathbf{Q}^{sub}}, 0))\})$ is added to the local moments for $\mathbf{Q}^{sub}$.

Computing and evaluating a local expansion inside $\mathbf{Q}^{sub}$ due to the contribution of $\mathbf{R}^{sub}$: one could iterate over each reference point $\mathbf{r}_{\mathbf{j_n}} \in \mathbf{R}^{sub}$ and compute the local moments $N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})$ due to $\mathbf{R}^{sub}$ up to $\mathcal{O}(p^D)$ terms (see Figure 20 and Figure 21). The *local accumulation* of the contribution of the reference node R requires $\mathcal{O}(|\mathbf{R}^{sub}|p^D)$ operations, and evaluating the *local expansion* for each $q_{i_m} \in \mathbf{Q}^{sub}$ requires a total of $\mathcal{O}(|\mathbf{Q}^{sub}|p^D)$ operations.

Converting far-field moments of $\mathbf{R}^{sub}$ to a local expansion of $\mathbf{Q}^{sub}$: Suppose $\mathbf{R}^{sub}$ has pre-computed far-field moments up to p^D terms. From the far-field moments, we can approximate the *local moments* of R but with some amount of error. This can be seen as a generalization of centroid-based approximation. [84] describes this method as one of the *translation operators*, called *far-field to local translation operator*:

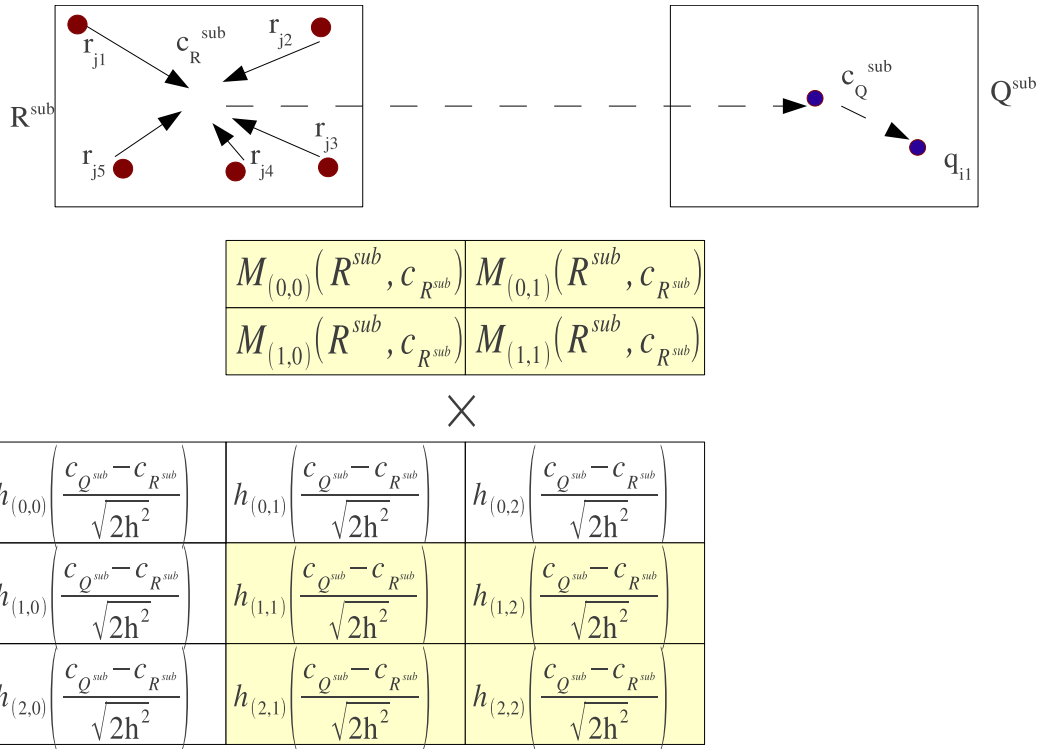


Figure 22: Two-dimensional far-field coefficients truncated after the first two terms in each dimension can be converted into a set of local moments using Equation (3.1.13). Computing $\widetilde{N}_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, 1))\})$ involves summing up the element-wise product between the matrix (or tensor in higher dimensions) consisting of the far-field moments and the two-by-two window over the Hermite functions whose upper left multi-index is $\boldsymbol{\beta}$. This figure shows how to compute $\widetilde{N}_{(1,1)}(\{(M(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, 1))\})$.

Lemma 3.1.2. Far-field to local (F2L) translation operator for Gaussian kernel *(Lemma 2.2 in [84])***:** *Given a reference node* $\mathbf{R}^{sub}$*, a query node* $\mathbf{Q}^{sub}$*, and the truncated far-field expansion centered at* $\mathbf{c}_{\mathbf{R}^{sub}}$ *of* $\mathbf{R}^{sub}$ *up to* $\mathcal{O}(p^D)$ *terms:*

$$\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right),$$

the Taylor expansion of the far-field expansion at $\mathbf{c}_{\mathbf{Q}^{sub}}$ *in* Q *is given by:*

$$\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{\boldsymbol{\beta} \geq 0} \widetilde{N}_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) \left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \text{ where}$$

for $\mathbf{q}_{\mathbf{i_m}} \in \mathbf{Q}^{sub}$,

$$\widetilde{N}_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) = \frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!} \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right) \tag{3.1.13}$$

Proof. Replacing the Hermite function portion of the expansion with its Taylor series:

$$\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$$

$$= \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \sum_{\boldsymbol{\beta} \geq 0} \frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!} h_{\boldsymbol{\alpha}+\boldsymbol{\beta}} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right) \left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$$

$$= \sum_{\boldsymbol{\beta} \geq 0} \left[\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!} \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}+\boldsymbol{\beta}} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right) \right] \left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$$

□

However, note $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\})$ has an infinite number of terms, and must be truncated after $\mathcal{O}(p^D)$ terms. In other words, the local moments accumulated for $\mathbf{Q}^{sub}$ are the coefficients for $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \{(\mathbf{R}^{sub}, \mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) = \sum_{\boldsymbol{\beta} \leq p} \widetilde{N}_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) \left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$ (see Figure 22). Computing $\{\widetilde{N}_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta} \leq p}$ requires iterating over all of $\mathcal{O}(p^D)$ far-field moments $\{M_{\boldsymbol{\alpha}}(R, \mathbf{c}_{\mathbf{R}^{sub}})\}_{\boldsymbol{\alpha} \leq p}$ for each $L_{\boldsymbol{\beta}}(\{(R, T(\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})$. This operation is $\mathcal{O}(Dp^{2D})$.

These approximations are generally valid under certain conditions which depend on how the error bounds associated with these approximation methods are derived. Moreover, we have not discussed how to choose the method of approximation given a query and reference node pair and the order of approximation, i.e. the number of terms required to achieve a given level of error. We discuss the details in Section 3.1.5.

3.1.5 Truncation Error Bounds

Because the far-field and the local expansions are truncated after taking $\mathcal{O}(p^D)$ terms, we incur an error in approximation. The original error bounds for the Gaussian kernel in [84] were wrong and corrections were shown in [11]. Here we present the error bounds for 1) evaluating a truncated far-field expansion of a reference node for any query point $\mathbf{q} \in \mathbb{R}^D$; 2) evaluating a truncated local expansion of $\mathbf{Q}^{sub}$ due to the contribution of a reference node $\mathbf{R}^{sub}$ for any query point $\mathbf{q}_{\mathbf{i_m}} \in \mathbf{Q}^{sub}$; 3) evaluating a truncated local expansion formed from converting a truncated far-field expansion of

a reference node $\mathbf{R}^{sub}$ for any query point $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$. Note that these error bounds place restrictions on the size of the nodes in consideration: reference node, query node, or both. First we start with the truncation error bound for evaluating the far-field expansion formed for a given reference node.

Lemma 3.1.3. Error bound for evaluating a truncated far-field expansion *[11]***:** *Suppose we are given a far-field expansion of* $\mathbf{R}^{sub}$ *about its centroid* $\mathbf{c}_{\mathbf{R}^{sub}}$*:* $\widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$ *where* $M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}}$*. If* $\forall \mathbf{r_{j_n}} \in \mathbf{R}^{sub}$ *satisfies* $||\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}||_\infty < rh$ *for* $r < 1$*, then for any* $\mathbf{q} \in \mathbb{R}^D$*,*

$$\left| \widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) - \Phi(\mathbf{q}; \mathbf{R}^{sub}) \right| \leq \frac{|\mathbf{R}^{sub}|}{(1-r)^D} \sum_{k=0}^{D-1} \binom{D}{k} (1 - r^{p+1})^k \left(\frac{r^{p+1}}{\sqrt{(p+1)!}} \right)^{D-k} \tag{3.1.14}$$

Proof. We expand the far-field expansion as a product of one-dimensional Hermite functions and utilize a bound on one-dimensional Hermite functions due to [184]: $\frac{1}{n!}|h_n(x)| \leq \frac{2^{\frac{n}{2}}}{\sqrt{n!}} \exp\left(\frac{-x^2}{2}\right), n \geq 0, x \in \mathbb{R}^1$.

$$u_{p_d}(\mathbf{q}[d], \mathbf{r_{j_n}}[d], \mathbf{c}_{\mathbf{R}^{sub}}[d]) = \sum_{n_i=0}^{p} \frac{1}{n_i!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)^{n_i} h_{n_i} \left(\frac{\mathbf{q}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)$$

$$v_{p_d}(\mathbf{q}[d], \mathbf{r_{j_n}}[d], \mathbf{c}_{\mathbf{R}^{sub}}[d]) = \sum_{n_i=p+1}^{\infty} \frac{1}{n_i!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)^{n_i} h_{n_i} \left(\frac{\mathbf{q}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)$$

$$\exp\left(\frac{-||\mathbf{q} - \mathbf{r_{j_n}}||^2}{2h^2} \right) = \prod_{d=1}^{D} \left(u_{p_d}(\mathbf{q}[d], \mathbf{r_{j_n}}[d], \mathbf{c}_{\mathbf{R}^{sub}}[d]) + v_{p_d}(\mathbf{q}[d], \mathbf{r_{j_n}}[d], \mathbf{c}_{\mathbf{R}^{sub}}[d]) \right)$$

We obtain for $1 \leq d \leq D$:

$$u_{p_d}(\mathbf{q}[d], \mathbf{r}_{\mathbf{j_n}}[d], \mathbf{c}_{\mathbf{R}^{sub}}[d]) \leq \sum_{n_i=0}^{p} \frac{1}{n_i!} \left| \frac{\mathbf{r}_{\mathbf{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right|^{n_i} \left| h_{n_i} \left(\frac{\mathbf{q}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \right|$$

$$\leq \sum_{n_i=0}^{p} \left| \frac{rh}{\sqrt{2h^2}} \right|^{n_i} \frac{2^{\frac{n_i}{2}}}{\sqrt{n_i!}} \exp\left(-\frac{(\mathbf{q}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d])^2}{4h^2} \right) \leq \sum_{n_i=0}^{p} r^{n_i} \leq \frac{1 - r^{p+1}}{1 - r}$$

$$v_{p_d}(\mathbf{q}[d], \mathbf{r}_{\mathbf{j_n}}[d], \mathbf{c}_{\mathbf{R}^{sub}}[d]) \leq \sum_{n_i=p+1}^{\infty} \frac{1}{n_i!} \left| \frac{\mathbf{r}_{\mathbf{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right|^{n_i} \left| h_{n_i} \left(\frac{\mathbf{q}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \right|$$

$$\leq \sum_{n_i=p+1}^{\infty} \left| \frac{rh}{\sqrt{2h^2}} \right|^{n_i} \frac{2^{\frac{n_i}{2}}}{\sqrt{n_i!}} \exp\left(-\frac{(\mathbf{q}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d])^2}{4h^2} \right) \leq \frac{1}{\sqrt{(p+1)!}} \sum_{n_i=p+1}^{\infty} r^{n_i} \leq \frac{1}{\sqrt{(p+1)!}} \frac{r^{p+1}}{1-r}$$

Therefore,

$$\left| \prod_{d=1}^{D} u_{p_d}(\mathbf{q}[d], \mathbf{r}_{\mathbf{j_n}}[d], \mathbf{c}_{\mathbf{R}^{sub}}[d]) - \exp\left(\frac{-||\mathbf{q} - \mathbf{r}_{\mathbf{j_n}}||^2}{2h^2} \right) \right|$$

$$\leq (1-r)^{-D} \sum_{k=0}^{D-1} \binom{D}{k} (1 - r^{p+1})^k \left(\frac{r^{p+1}}{\sqrt{(p+1)!}} \right)^{D-k}$$

$$\left| \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right) - \sum_{r_{j_n} \in R} \exp\left(\frac{-||\mathbf{q} - \mathbf{r}_{\mathbf{j_n}}||^2}{2h^2} \right) \right|$$

$$\leq \frac{|\mathbf{R}^{sub}|}{(1-r)^D} \sum_{k=0}^{D-1} \binom{D}{k} (1 - r^{p+1})^k \left(\frac{r^{p+1}}{\sqrt{(p+1)!}} \right)^{D-k}$$

□

Intuitively, this theorem implies that evaluating a truncated far-field expansion for a query point (regardless of its location) requires that the reference points used to form the expansion are within the bandwidth h in each dimension from the centroid $\mathbf{c}_{\mathbf{R}^{sub}}$ (i.e. the reference node has a maximum side length of $2h$).

The following gives the truncation bound for the local expansion formed inside a query node whose bound is within a hypercube of some side length.

Lemma 3.1.4. Error bound for evaluating a truncated local expansion: *Suppose we are given the local expansion about* $\mathbf{c}_{\mathbf{Q}^{sub}}$ *of the given query node* $\mathbf{Q}^{sub}$ *accounting for the kernel sum contribution of* $\mathbf{R}^{sub}$*:* $\widetilde{\Phi}(q_{i_m}; \{(\mathbf{R}^{sub}, \mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) =$

$\sum_{\boldsymbol{\beta}\leq p} N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub},(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$ *where* $\mathbf{q}_{\mathbf{i_m}}\in\mathbf{Q}^{sub}$ *and* $N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub},(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})=\sum_{\mathbf{r_{j_n}}\in\mathbf{R}^{sub}}\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}h_{\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}-r_{j_n}}{\sqrt{2h^2}}\right)$

If $\forall\mathbf{q}_{\mathbf{i_m}}\in\mathbf{Q}^{sub}$ *satisfies* $||\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}||_\infty<rh$ *for* $r<1$*, then for any* $\mathbf{q}_{\mathbf{i_m}}\in\mathbf{Q}^{sub}$*:*

$$\left|\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(\mathbf{R}^{sub},\mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})-\Phi(\mathbf{q}_{\mathbf{i_m}};\mathbf{R}^{sub})\right|\leq\frac{|\mathbf{R}^{sub}|}{(1-r)^D}\sum_{k=0}^{D-1}\binom{D}{k}(1-r^p)^k\left(\frac{r^p}{\sqrt{p!}}\right)^{D-k} \tag{3.1.15}$$

Proof. Taylor expansion of the Hermite function yields:

$$\begin{aligned}
&\exp\left(\frac{-||\mathbf{q}_{\mathbf{i_m}}-\mathbf{r}_{\mathbf{j_n}}||^2}{2h^2}\right)\\
=&\sum_{\boldsymbol{\beta}\geq 0}\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}\sum_{\boldsymbol{\alpha}\geq 0}\frac{1}{\boldsymbol{\alpha}!}\left(\frac{\mathbf{r}_{\mathbf{j_n}}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}}h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}\\
=&\sum_{\boldsymbol{\beta}\geq 0}\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}\sum_{\boldsymbol{\alpha}\geq 0}\frac{1}{\boldsymbol{\alpha}!}\left(\frac{\mathbf{c}_{\mathbf{R}^{sub}}-\mathbf{r}_{\mathbf{j_n}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}}(-1)^{|\boldsymbol{\alpha}|}h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}\\
=&\sum_{\boldsymbol{\beta}\geq 0}\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}h_{\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}-\mathbf{r}_{\mathbf{j_n}}}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}
\end{aligned}$$

Use $\exp\left(\frac{-||\mathbf{q}_{\mathbf{i_m}}-\mathbf{r}_{\mathbf{j_n}}||^2}{2h^2}\right)=\prod_{d=1}^{D}\left(u_p(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d])+v_p(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d])\right)$ for $1\leq d\leq D$, where

$$u_{p_d}(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d])=\sum_{n_d=0}^{p}\frac{(-1)^{n_d}}{n_d!}h_{n_d}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d]-\mathbf{r}_{\mathbf{j_n}}[d]}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q}_{\mathbf{m_i}}[d]-\mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{n_d}$$

$$v_{p_d}(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d])=\sum_{n_i=p+1}^{\infty}\frac{(-1)^{n_d}}{n_d!}h_{n_d}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d]-\mathbf{r}_{\mathbf{j_n}}[d]}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q}_{\mathbf{m_i}}[d]-\mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{n_d}$$

These univariate functions respectively satisfy $u_{p_d}(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d])\leq\frac{1-r^{p+1}}{1-r}$ and $v_{p_d}(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d])\leq\frac{1}{\sqrt{(p+1)!}}\frac{r^{p+1}}{1-r}$, for $1\leq d\leq D$, achieving the multivariate bound. The proof is similar as in the one given in Lemma 3.1.3. □

Lastly, we present the error bound for evaluating a truncated local expansion formed from a truncated far-field expansion, which requires that both the query node and the reference node are "small":

Lemma 3.1.5. Error bound for evaluating a truncated local expansion converted from an already truncated far-field expansion: *A truncated far-field expansion centered about the centroid* $\mathbf{c}_{\mathbf{R}^{sub}}$ *of* $\mathbf{R}^{sub}$*,*

$$\widetilde{\Phi}(\mathbf{q};\{(\mathbf{R}^{sub},\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}},p))\})=\sum_{\boldsymbol{\alpha}\leq p}M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}})h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{q}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)$$

has the following local expansion about $\mathbf{c}_{\mathbf{Q}^{sub}}$ *of* $\mathbf{Q}^{sub}$ *for* $\mathbf{q}_{\mathbf{i_m}}\in\mathbf{Q}^{sub}$*:* $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(\mathbf{R}^{sub},\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}},p))\})=\sum_{\boldsymbol{\beta}\geq 0}N_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$ *where:* $N_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})=\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}\sum_{\boldsymbol{\alpha}\leq p}M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}})h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)$*. Let* $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(M(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})=\sum_{\boldsymbol{\beta}\leq p}\widetilde{N}_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$*, a truncation of the local expansion of* $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(\mathbf{R}^{sub},\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}},p))\})$ *after* $\mathcal{O}(p^D)$ *terms.*

If $\forall\mathbf{q}_{\mathbf{i_m}}\in\mathbf{Q}^{sub}$ *satisfies* $||\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}||_\infty<rh$ *and* $\forall\mathbf{r}_{\mathbf{j_n}}\in\mathbf{R}^{sub}$ *satisfies* $||\mathbf{r}_{\mathbf{j_n}}-\mathbf{c}_{\mathbf{R}^{sub}}||_\infty<rh$ *for* $r<\frac{1}{2}$*, then for any* $\mathbf{q}_{\mathbf{i_m}}\in\mathbf{Q}^{sub}$*:*

$$\begin{aligned}&\left|\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(M(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})-\Phi(\mathbf{q}_{\mathbf{i_m}};\mathbf{R}^{sub})\right|\\ \leq&\frac{|\mathbf{R}^{sub}|}{(1-2r)^{2D}}\sum_{k=0}^{D-1}\binom{D}{k}((1-(2r)^p)^2)^k\left(\frac{((2r)^p)(2-(2r)^p)}{\sqrt{p!}}\right)^{D-k}\end{aligned}\tag{3.1.16}$$

Proof. We define for $1\leq d\leq D$:

$$\begin{aligned}u_{p_d}&=u_p(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d],\mathbf{c}_{\mathbf{R}^{sub}}[d])\\ v_{p_d}&=v_p(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d],\mathbf{c}_{\mathbf{R}^{sub}}[d])\\ w_{p_d}&=w_p(\mathbf{q}_{\mathbf{i_m}}[d],\mathbf{r}_{\mathbf{j_n}}[d],\mathbf{c}_{\mathbf{Q}^{sub}}[d],\mathbf{c}_{\mathbf{R}^{sub}}[d])\end{aligned}$$

$$
\begin{aligned}
u_{p_d} = & \sum_{n_i=0}^{p} \frac{(-1)^{n_i}}{n_i!} \sum_{n_j=0}^{p} \frac{1}{n_j!} \left(\frac{\mathbf{c}_{\mathbf{R}^{sub}}[d] - \mathbf{r}_{\mathbf{j_n}}[d]}{\sqrt{2h^2}} \right)^{n_j} (-1)^{n_j} \\
& h_{n_i+n_j} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \left(\frac{\mathbf{q}_{\mathbf{i_m}}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}} \right)^{n_i} \\
v_{p_d} = & \sum_{n_i=0}^{p} \frac{(-1)^{n_i}}{n_i!} \sum_{n_j=p+1}^{\infty} \frac{1}{n_j!} \left(\frac{\mathbf{c}_{\mathbf{R}^{sub}}[d] - \mathbf{r}_{\mathbf{j_n}}[d]}{\sqrt{2h^2}} \right)^{n_j} (-1)^{n_j} \\
& h_{n_i+n_j} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \left(\frac{\mathbf{q}_{\mathbf{i_m}}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}} \right)^{n_i} \\
w_{p_d} = & \sum_{n_i=p+1}^{\infty} \frac{(-1)^{n_i}}{n_i!} \sum_{n_j=0}^{\infty} \frac{1}{n_j!} \left(\frac{\mathbf{c}_{\mathbf{R}^{sub}}[d] - \mathbf{r}_{\mathbf{j_n}}[d]}{\sqrt{2h^2}} \right)^{n_j} (-1)^{n_j} \\
& h_{n_i+n_j} \left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \left(\frac{\mathbf{q}_{\mathbf{i_m}}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}} \right)^{n_i}
\end{aligned}
$$

Note that $\exp\left(\frac{-||\mathbf{q}_{\mathbf{i_m}} - \mathbf{r}_{\mathbf{j_n}}||^2}{2h^2}\right) = \prod_{d=1}^{D} (u_{p_d} + v_{p_d} + w_{p_d})$ for $1 \leq d \leq D$. Using the bound for Hermite functions and the property of geometric series, we obtain:

$$
\begin{aligned}
u_{p_d} &\leq \sum_{n_i=0}^{p} \sum_{n_j=0}^{p} (2r)^{n_i} (2r)^{n_j} = \left(\frac{1 - (2r)^{p+1})}{1 - 2r} \right)^2 \\
v_{p_d} &\leq \frac{1}{\sqrt{(p+1)!}} \sum_{n_i=0}^{p} \sum_{n_j=p+1}^{\infty} (2r)^{n_i} (2r)^{n_j} = \frac{1}{\sqrt{(p+1)!}} \left(\frac{1 - (2r)^{p+1}}{1 - 2r} \right) \left(\frac{(2r)^{p+1}}{1 - 2r} \right) \\
w_{p_d} &\leq \frac{1}{\sqrt{(p+1)!}} \sum_{n_i=p+1}^{\infty} \sum_{n_j=0}^{\infty} (2r)^{n_i} (2r)^{n_j} = \frac{1}{\sqrt{(p+1)!}} \left(\frac{1}{1 - 2r} \right) \left(\frac{(2r)^{p+1}}{1 - 2r} \right)
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
& \left| \prod_{d=1}^{D} u_{p_d} - \exp\left(\frac{-||\mathbf{q}_{\mathbf{i_m}} - \mathbf{r}_{\mathbf{j_n}}||^2}{2h^2} \right) \right| \\
\leq & (1 - 2r)^{-2D} \sum_{k=0}^{D-1} \binom{D}{k} ((1 - (2r)^{p+1})^2)^k \left(\frac{((2r)^{p+1})(2 - (2r)^{p+1})}{\sqrt{(p+1)!}} \right)^{D-k} \\
& \left| \widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \{(M(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), \widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}}, p))\}) - \Phi(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}^{sub}) \right| \\
\leq & \frac{|\mathbf{R}^{sub}|}{(1 - 2r)^{2D}} \sum_{k=0}^{D-1} \binom{D}{k} ((1 - (2r)^{p+1})^2)^k \left(\frac{((2r)^{p+1})(2 - (2r)^{p+1})}{\sqrt{(p+1)!}} \right)^{D-k}
\end{aligned}
$$

□

[183] proposes an interesting idea of using Stirling's formula (for any non-negative

Algorithm 3.1.1 FARFIELDORDER($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \tau$): Determines the order of approximation needed for evaluating a far-field expansion of the reference node $\mathbf{R}^{sub}$.

$r \leftarrow$ the widest length of the bounding box of $\mathbf{R}^{sub}$ divided by $2h$.
if $r \geq 1$ **then**
  return ∞
else
  $p \leftarrow 0$
  while $p \leq p_{max}$ **do**
    if $\frac{|\mathbf{R}^{sub}|}{(1-r)^D} \sum_{k=0}^{D-1} \binom{D}{k} (1 - r^{p+1})^k \left(\frac{r^{p+1}}{\sqrt{(p+1)!}} \right)^{D-k} \leq \tau$ **then**
      return p
    $p \leftarrow p + 1$
  return ∞

integer n, $\left(\frac{n+1}{e}\right)^n \leq n!$) to lift the node size constraint. This could allow approximation of larger regions that possibly contain more points. Unfortunately, the error bounds derived in [183] were also incorrect. We have derived the necessary corrected error bounds based on the techniques in [11]. However, we do not include the derivations here since using these bounds actually degraded performance in our algorithm.

3.1.6 Determining the Approximation Order

Note that Lemma 3.1.3, Lemma 3.1.4, and Lemma 3.1.5 upper-bound the approximation error $\left|\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{sub}) - \Phi(\mathbf{q}, \mathbf{R}^{sub})\right|$ given that we use $\mathcal{O}(p^D)$ terms in the appropriate expansion type. Nevertheless, all three lemmas can be re-phrased to answer the question in reverse: given the maximum user-desired absolute error, what is the order of approximation required to achieve it? This question rises naturally within our dual-tree based algorithm that bounds the kernel sum approximation error on each part in a partition of the reference dataset $\mathbf{R}$.

Algorithm 3.1.1 shows how to determine the necessary order of the far-field expansion for the given reference node $\mathbf{R}^{sub}$ such that $\left|\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{sub}) - \Phi(\mathbf{q}, \mathbf{R}^{sub})\right| \leq \tau$. That is, the approximation error due to the far-field expansion of $\mathbf{R}^{sub}$ is bounded by the error allocated for approximating the contribution of $\mathbf{R}^{sub}$. Using far-field expansion

Algorithm 3.1.2 LocalAccumulationOrder($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \tau$): Determining the order of approximation needed for forming a local expansion of the contribution from the given reference node $\mathbf{R}^{sub}$ for the query node $\mathbf{Q}^{sub}$.

$r \leftarrow$ the widest length of the bounding box of $\mathbf{Q}^{sub}$ divided by $2h$.
if $r \geq 1$ **then**
  return ∞
else
  $p \leftarrow 0$
  while $p \leq p_{max}$ **do**
    if $\frac{|\mathbf{R}^{sub}|}{(1-r)^D} \sum_{k=0}^{D-1} \binom{D}{k} (1 - r^{p+1})^k \left(\frac{r^{p+1}}{\sqrt{(p+1)!}} \right)^{D-k} \leq \tau$ **then**
      return p
    $p \leftarrow p + 1$
  return ∞

based approximation requires a "small" reference node. The algorithm computes the ratio of the maximum side length of $\mathbf{R}^{sub}$ to twice the bandwidth h and determines the least order required for achieving the maximum absolute error τ by evaluating the right-hand side of Equation (3.1.14) iteratively on different values of p.

Algorithm 3.1.2 shows how to determine the necessary order of the local expansion formed by directly accumulating the contribution of the given reference node $\mathbf{R}^{sub}$ onto the given query node $\mathbf{Q}^{sub}$. This approximation method requires the query node $\mathbf{Q}^{sub}$ to have the maximum side length within twice the bandwidth. The algorithm determines the least order required by using the right-hand side of Equation (3.1.15).

Finally, Algorithm 3.1.3 determines the necessary order of local expansion formed by converting a truncated far-field expansion of the given reference node $\mathbf{R}^{sub}$. In contrast to the two previous algorithms, this one requires both the query node $\mathbf{Q}^{sub}$ and the reference node $\mathbf{R}^{sub}$ to have a maximum side length less than the bandwidth h. After the node size requirements are satisfied, the least order required for achieving τ absolute error is obtained by using the right-hand side of Equation (3.1.16).

Algorithm 3.1.3 CONVERTFARFIELDTOLOCALORDER($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \tau$): Determining the order of approximation needed for evaluating a far-field expansion of the given reference node $\mathbf{R}^{sub}$.

$r_1 \leftarrow$ the widest length of the bounding box of $\mathbf{Q}^{sub}$ divided by $4h$.
$r_2 \leftarrow$ the widest length of the bounding box of $\mathbf{R}^{sub}$ divided by $4h$.
$r \leftarrow \max\{r_1, r_2\}$
if $r \geq \frac{1}{2}$ **then**
  return ∞
else
  $p \leftarrow 0$
  while $p \leq p_{max}$ **do**
    if $\frac{|\mathbf{R}^{sub}|}{(1-2r)^{2D}} \sum_{k=0}^{D-1} \binom{D}{k} ((1-(2r)^{p+1})^2)^k \left(\frac{((2r)^{p+1})(2-(2r)^{p+1})}{\sqrt{(p+1)!}} \right)^{D-k} \leq \tau$ **then**
      return p
    $p \leftarrow p+1$
  return ∞

3.1.7 Deriving the Hierarchical FGT

Until now, we have discussed the approximation methods developed for a non-hierarchical version of fast Gauss transform described in [84]. In this section, we derive the two additional translation operators that extend the original fast Gauss transform to use a hierarchical data structure. Here we consider the reference tree, which enables the consideration of the different portions of the reference set $\mathbf{R}$ at a different granularity. Given the computed far-field moments of $\mathbf{R}^{sub,L}$ and $\mathbf{R}^{sub,R}$, each centered at $\mathbf{c}_{\mathbf{R}^{sub,L}}$ and $\mathbf{c}_{\mathbf{R}^{sub,R}}$, how can we efficiently compute the far-field moments of $\mathbf{R}^{sub}$ centered at $\mathbf{c}_{\mathbf{R}^{sub}}$, the parent of $\mathbf{R}^{sub,L}$ and $\mathbf{R}^{sub,R}$? The first operator allows the efficient bottom-up pre-computation of the Hermite moments in the reference tree.

Lemma 3.1.6. Shifting a far-field expansion of a reference node to a new center (F2F translation operator for the Gaussian kernel): *Given the far-field expansion centered at* $\mathbf{c}_{\mathbf{R}^{sub}}$ *in* $\mathbf{R}^{sub}$*:*

$$\widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$$

This same far-field expansion shifted to a new location $\mathbf{c}'$ *is given by:*

$$\widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}', p))\}) = \sum_{\boldsymbol{\gamma} \geq 0} M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub}, \mathbf{c}') h_{\boldsymbol{\gamma}} \left(\frac{\mathbf{q} - \mathbf{c}'}{\sqrt{2h^2}} \right)$$

where

$$M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub}, \mathbf{c}') = \sum_{0 \leq \boldsymbol{\alpha} \leq \boldsymbol{\gamma}} \frac{1}{(\boldsymbol{\gamma} - \boldsymbol{\alpha})!} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \left(\frac{\mathbf{c}_{\mathbf{R}^{sub}} - \mathbf{c}'}{\sqrt{2h^2}} \right)^{\boldsymbol{\gamma} - \boldsymbol{\alpha}} \tag{3.1.17}$$

Proof. Replace the Hermite part of the expansion by a new Taylor series:

$$\begin{aligned}
&\widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right) \\
&= \sum_{\boldsymbol{\alpha} \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \sum_{\boldsymbol{\beta} \geq 0} \frac{1}{\boldsymbol{\beta}!} \left(\frac{\mathbf{c}' - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}} (-1)^{|\boldsymbol{\beta}|} h_{\boldsymbol{\alpha}+\boldsymbol{\beta}} \left(\frac{\mathbf{q} - \mathbf{c}'}{\sqrt{2h^2}} \right) \\
&= \sum_{\boldsymbol{\alpha} \leq p} \sum_{\boldsymbol{\beta} \geq 0} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \frac{1}{\boldsymbol{\beta}!} \left(\frac{\mathbf{c}' - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}} (-1)^{|\boldsymbol{\beta}|} h_{\boldsymbol{\alpha}+\boldsymbol{\beta}} \left(\frac{\mathbf{q} - \mathbf{c}'}{\sqrt{2h^2}} \right) \\
&= \sum_{\boldsymbol{\alpha} \leq p} \sum_{\boldsymbol{\beta} \geq 0} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \frac{1}{\boldsymbol{\beta}!} \left(\frac{\mathbf{c}_{\mathbf{R}^{sub}} - \mathbf{c}'}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}} h_{\boldsymbol{\alpha}+\boldsymbol{\beta}} \left(\frac{\mathbf{q} - \mathbf{c}'}{\sqrt{2h^2}} \right) \\
&= \sum_{\boldsymbol{\gamma} \leq p} \left[\sum_{0 \leq \boldsymbol{\alpha} \leq \boldsymbol{\gamma}} \frac{1}{(\boldsymbol{\gamma} - \boldsymbol{\alpha})!} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \left(\frac{\mathbf{c}_{\mathbf{R}^{sub}} - \mathbf{c}'}{\sqrt{2h^2}} \right)^{\boldsymbol{\gamma} - \boldsymbol{\alpha}} \right] h_{\boldsymbol{\gamma}} \left(\frac{\mathbf{q} - \mathbf{c}'}{\sqrt{2h^2}} \right)
\end{aligned}$$

where $\boldsymbol{\gamma} = \boldsymbol{\alpha} + \boldsymbol{\beta}$. □

Using Lemma 3.1.6, we can compute the far-field moments of $\mathbf{R}^{sub}$ centered at $\mathbf{c}_{\mathbf{R}^{sub}}$ by translating the moments $\{M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,L}, c_{\mathbf{R}^{sub,L}})\}_{\boldsymbol{\gamma} \leq p}$ and $\{M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,R}, c_{\mathbf{R}^{sub,R}})\}_{\boldsymbol{\gamma} \leq p}$ to form the moments $\{M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,L}, \mathbf{c}_{\mathbf{R}^{sub}})\}_{\boldsymbol{\gamma} \leq p}$ and $\{M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,R}, \mathbf{c}_{\mathbf{R}^{sub}})\}_{\boldsymbol{\gamma} < p}$. The far-field moments of $\mathbf{R}^{sub} = \mathbf{R}^{sub,L} \cup \mathbf{R}^{sub,R}$ are $\{M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,L}, \mathbf{c}_{\mathbf{R}^{sub}}) + M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,R}, \mathbf{c}_{\mathbf{R}^{sub}})\}$ and

$$\widetilde{\Phi}(\mathbf{q}; \{(R, F(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{\boldsymbol{\gamma} \leq p} (M_{\boldsymbol{\gamma}}(R^L, \mathbf{c}_{\mathbf{R}^{sub}}) + M_{\boldsymbol{\gamma}}(R^R, \mathbf{c}_{\mathbf{R}^{sub}})) h_{\boldsymbol{\gamma}} \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$$

Computing each $M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,L}, \mathbf{c}_{\mathbf{R}^{sub}})$ from $M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,L}, \mathbf{c}_{\mathbf{R}^{sub,L}})$ (and each $M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,R}, \mathbf{c}_{\mathbf{R}^{sub}})$ from $M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub,R}, \mathbf{c}_{\mathbf{R}^{sub,R}})$) requires iterating over at most $\mathcal{O}(p^D)$ terms. This operation runs in $\mathcal{O}(Dp^{2D})$, which can be more efficient than computing the far-field moments of $\mathbf{R}^{sub}$ centered at $\mathbf{c}_{\mathbf{R}^{sub}}$ from scratch (which is $\mathcal{O}(|\mathbf{R}^{sub}|Dp^D)$).

The next translation operator acts as a "clean-up" routine in a hierarchical algorithm. Since we can approximate at different scales in the query tree, we must

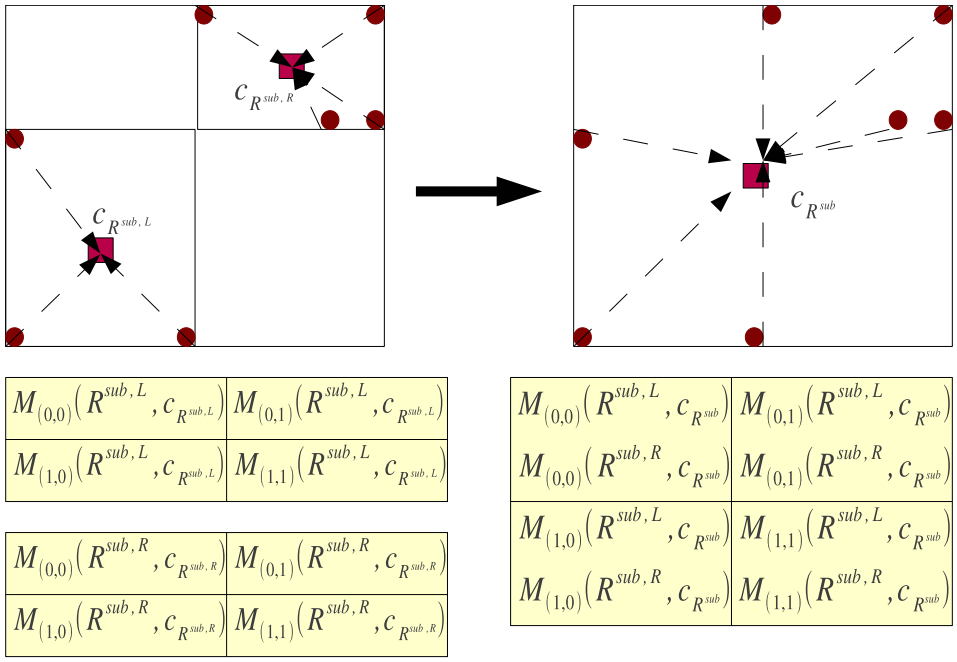


Figure 23: Given the far-field moments of $\mathbf{R}^{sub,L}$ and $\mathbf{R}^{sub,R}$ illustrated in the first two tables, Theorem 3.1.6 can re-center each set of far-field moments of $\mathbf{R}^{sub,L}$ and $\mathbf{R}^{sub,R}$ at centroid $\mathbf{c}_{\mathbf{R}^{sub}}$. The re-centered far-field moments are shown in the third table with two numbers, each contributed by $\mathbf{R}^{sub,L}$ and $\mathbf{R}^{sub,R}$. The far-field moments of $\mathbf{R}^{sub}$ are then computed by adding up the two re-centered moments entry-wise.

somehow combine all the approximations at the end of the computation. By performing a breadth-first traversal of the query tree, the L2L operator shifts a node's local expansion to the centroid of each child.

Lemma 3.1.7. Shifting a combined local expansion of a query node to a new center (L2L translation operator for Gaussian kernel): *Given a combined local expansion centered at* $\mathbf{c}_{\mathbf{Q}^{sub}}$ *of the given query node* $\mathbf{Q}^{sub}$*:*

$$\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) = \sum_{\boldsymbol{\beta} \leq p} \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$$

Shifting this local expansion to the new center $\mathbf{c}' \in \mathbf{Q}^{sub}$ *yields:*

$$\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))$$
$$= \sum_{\boldsymbol{\alpha} \leq p} \left[\sum_{\boldsymbol{\beta} \geq \boldsymbol{\alpha}} \frac{\boldsymbol{\beta}!}{\boldsymbol{\alpha}!(\boldsymbol{\beta} - \boldsymbol{\alpha})!} \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \left(\frac{\mathbf{c}' - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta} - \boldsymbol{\alpha}} \right] \left(\frac{\mathbf{q} - \mathbf{c}'}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}}$$

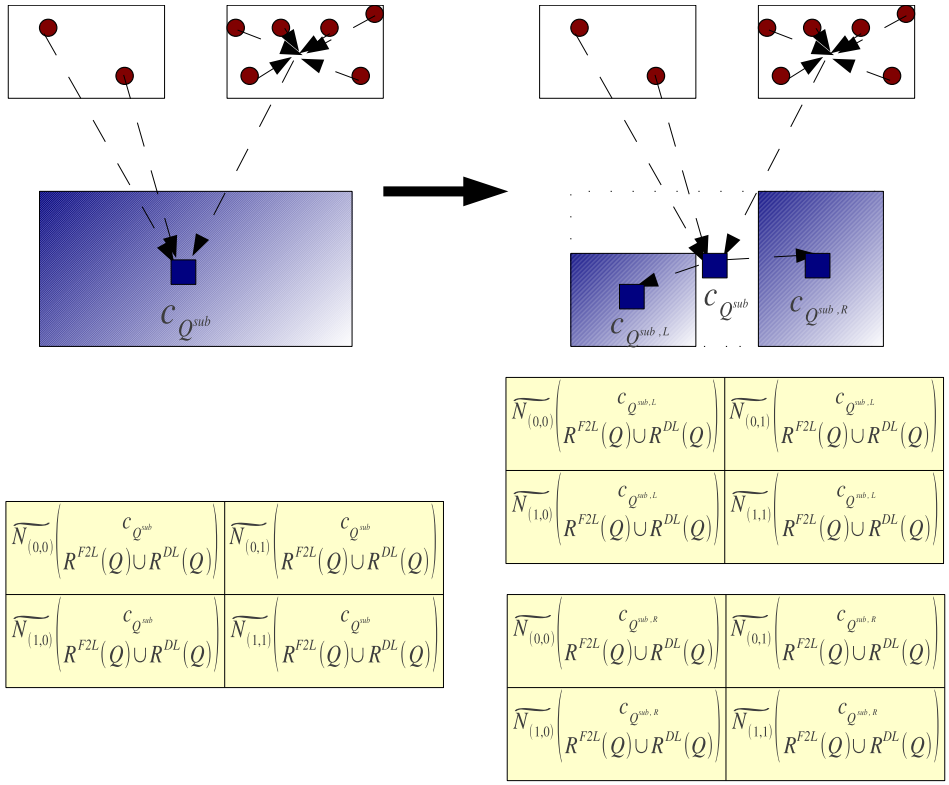


Figure 24: Given the local moments centered at $\mathbf{c}_{\mathbf{Q}^{sub}}$, Theorem 3.1.7 can re-center them at the two centroids $\mathbf{c}_{\mathbf{Q}^{sub,L}}$ and $\mathbf{c}_{\mathbf{Q}^{sub,R}}$.

where we denote

$$\begin{aligned}
&\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}', \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \\
&= \sum_{\boldsymbol{\beta} \geq \boldsymbol{\alpha}} \frac{\boldsymbol{\beta}!}{\boldsymbol{\alpha}!(\boldsymbol{\beta}-\boldsymbol{\alpha})!} \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \left(\frac{\mathbf{c}' - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}-\boldsymbol{\alpha}}
\end{aligned} \tag{3.1.18}$$

Proof. Use the multinomial theorem to expand about the new center c':

$$\begin{aligned}
&\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \\
&= \sum_{\boldsymbol{\beta} \leq p} \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}} \\
&= \sum_{\boldsymbol{\beta} \leq p} \sum_{\boldsymbol{\alpha} \leq \boldsymbol{\beta}} \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \frac{\boldsymbol{\beta}!}{\boldsymbol{\alpha}!(\boldsymbol{\beta}-\boldsymbol{\alpha})!} \left(\frac{\mathbf{c}' - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}-\boldsymbol{\alpha}} \left(\frac{\mathbf{q} - \mathbf{c}'}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}}
\end{aligned}$$

whose summation order can be interchanged to achieve the result. □

Using Lemma 3.1.7, we can shift the local moments of $\mathbf{Q}^{sub}$ centered at $\mathbf{c}_{\mathbf{Q}^{sub}}$ to a different expansion center, such as an expansion center of one of the child nodes of $\mathbf{Q}^{sub}$. Let p be the maximum approximation order used among the reference nodes pruned via far-to-local translation ($\mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})$) and direct local accumulation ($\mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub})$). The local moment propagation to both child nodes of $\mathbf{Q}^{sub}$ is achieved by the following operations:

$$\{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub,L}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub,L}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub,L}))\}_{\boldsymbol{\beta} \leq p}$$

$$\leftarrow \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub,L}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub,L}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub,L}))\}_{\boldsymbol{\beta} \leq p} + \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub,L}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \leq p}$$

$$\{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub,R}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub,R}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub,R}))\}_{\boldsymbol{\beta} \leq p}$$

$$\leftarrow \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub,R}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub,R}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub,R}))\}_{\boldsymbol{\beta} \leq p} + \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub,R}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \leq p}$$

where the addition operation is an element-wise operation over multi-index positions.

3.1.8 Choosing the Best Approximation Method

Suppose we are given a query node $\mathbf{Q}^{sub}$ and a reference node $\mathbf{R}^{sub}$ pair during the invocation of Algorithm 3.1.10. CanSummarize function for the higher-order DFGT algorithm has four approximation methods available: $\mathsf{A} \in \{\mathsf{D}, \widetilde{\mathsf{N}}(\mathbf{c}, p), \mathsf{F}(\mathbf{c}, p), \mathsf{N}(\mathbf{c}, p)\}$ (see Section 3.1.2). Because we would like to avoid exhaustive computations, the higher-order DFGT algorithm uses only three of the approximation methods and defers exhaustive computations until query/reference leaf pairs are encountered. Algorithm 3.1.4 tests whether the given query node and reference node pair can be approximated by evaluating the far-field moments of $\mathbf{R}^{sub}$, computing direct local accumulation due to $\mathbf{R}^{sub}$, and translating some of the terms that constitute the far-field moments of $\mathbf{R}^{sub}$ (far-field-to-local translation operator) and evaluates the asymptotic cost of each approximation. Algorithm 3.1.4 then determines the approximation method with the lowest asymptotic cost. This idea was originally introduced in [84] in the description of the original fast Gauss transform algorithm. The key

Algorithm 3.1.4 ChooseBestMethod($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \tau$): Chooses the FMM-type approximation with the least cost for a query and reference node pair.

$p_F \leftarrow$ FarFieldOrder($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \tau$)
$p_D \leftarrow$ LocalAccumulationOrder($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \tau$)
$p_T \leftarrow$ ConvertFarFieldToLocalOrder($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \tau$)
$c_F \leftarrow |\mathbf{Q}^{sub}| D^{p_F+1}$, $c_D \leftarrow |\mathbf{R}^{sub}| D^{p_D+1}$, $c_T \leftarrow D^{2p_T+1}$, $c_E \leftarrow D|\mathbf{Q}^{sub}||\mathbf{R}^{sub}|$
if $c_F = \min\{c_F, c_D, c_T, c_E\}$ **then**
  return $\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p_F)$
else if $c_D = \min\{c_F, c_D, c_T, c_E\}$ **then**
  return $\mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}}, p_D)$
else if $c_T = \min\{c_F, c_D, c_T, c_E\}$ **then**
  return $\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}}, p_T)$
else
  return D

difference is that even if Algorithm 3.1.4 returns D (when none of the other approximation methods can beat the cost of the exhaustive method), our hierarchical algorithm will not default to exhaustive evaluations and will consider the query points and reference points at a finer granularity, as shown in Algorithm 3.1.10.

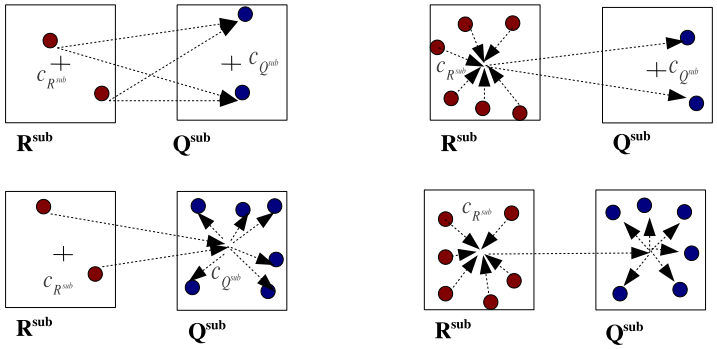


Figure 25: Four ways of approximating the contribution of a reference node to a query node. **Top left:** exhaustive computations (few reference/few query points); **Top right:** far-field moment evaluating (many reference/few query points); **Bottom left:** direct local moment accumulation (few reference/many query points); **Bottom right:** far-field-to-local translation (many reference/many query points).

3.1.9 Hierarchical FGT

Given the analytical machinery developed in the previous section, we now describe how to extend the centroid-based dual-tree [80, 82] to do higher-order approximations.

Algorithm 3.1.5 DFGTMAIN($\mathbf{Q}, \mathbf{R}$): The main KDE routine.

BUILDKDTREE($\mathbf{Q}$), BUILDKDTREE($\mathbf{R}$)
DFGTINITQ($\mathbf{Q}$), DFGTINITR($\mathbf{R}$), DFGT($\mathbf{Q}, \mathbf{R}$), DFGTPOST($\mathbf{Q}$)

The main structure of the algorithm is shown in Algorithm 3.1.5. We provide only a high-level overview of our algorithm and defer implementation details to Appendix.

Initialization of the query tree. Each query node maintains a vector storing $(p_{max}+1)^D$ terms, where p_{max} is a pre-determined limit on the approximation order[2] depending on the dimensionality of the query set $\mathbf{Q}$ and the reference set $\mathbf{R}$. For the experimental results, we have fixed $p_{max} = 5$ for $D = 2$, $p_{max} = 3$ for $D = 3$, $p_{max} = 1$ for $D = 4$ and $D = 5$, $p_{max} = 0$ for $D \geq 6$.

Algorithm 3.1.6 DFGTINITQ($\mathbf{Q}^{sub}$): Initializes query bound summary statistics.

{Initialize the node bound summary statistics.}
$\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow 0, \quad \Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow |\mathbf{R}|, \quad \Delta^l(\mathbf{Q}^{sub}) \leftarrow 0, \quad \Delta^u(\mathbf{Q}^{sub}) \leftarrow 0$
{Initialize translated local moments to be a vector of length $(p_{max}+1)^D$.}
$\widetilde{N}_{0 \leq i < (p_{max}+1)^D}(\mathbf{Q}^{sub}) \leftarrow 0$
if $\mathbf{Q}^{sub}$ is a leaf node **then**
  {Initialize for each query point.}
  for each $\mathbf{q}_{\mathbf{i_m}} \in \mathbf{Q}^{sub}$ **do**
    $\Phi^l(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}) \leftarrow 0, \qquad \Phi^u(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}) \leftarrow |\mathbf{R}|$
    $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}) \leftarrow 0, \qquad \widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}^{\mathsf{E}}(\mathbf{q}_{\mathbf{i_m}})) \leftarrow 0$
    $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}^{\mathsf{DF}}(\mathbf{q}_{\mathbf{i_m}})) \leftarrow 0, \qquad \widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}^{\mathsf{DL}}(\mathbf{q}_{\mathbf{i_m}}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{q}_{\mathbf{i_m}})) \leftarrow 0$
else
  DFGTINITQ($\mathbf{Q}^{sub,L}$), DFGTINITQ($\mathbf{Q}^{sub,R}$)

Pre-computation of far-field moments. Before the main KDE computation can begin, we pre-compute the far-field moments of each reference node in the reference tree up to $(p_{max}+1)^D$ terms. We show how to efficiently pre-compute the far-field moments of each reference node in the reference tree in Algorithm 3.1.7. The algorithm uses Equation (3.1.11) for the leaf node and Equation (3.1.17) for translating

[2]We impose this limit because the number of terms scales exponentially with the dimensionality D, $\mathcal{O}(p^D)$.

Algorithm 3.1.7 DFGTInitR($\mathbf{R}^{sub}$): Pre-computes far-field moments.

{Initialize the far-field moments of $\mathbf{R}^{sub}$ to be empty.}
for $i = 0$ to $i < (p_{max} + 1)^D$ **do**
  $M_{\text{PositionToMultiindex}(i,p_{max})}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \leftarrow 0$
if R is a leaf node **then**
  {Accumulate far-field moment from each point (Equation (3.1.11)).}
  AccumulateFarFieldMoment($\mathbf{R}^{sub}$)
else
  {Recursively compute the moments of the child nodes and combine them.}
  DFGTInitR($\mathbf{R}^{sub,L}$), DFGTInitR($\mathbf{R}^{sub,R}$)
  TransFarToFar($\mathbf{R}^{sub,L}, \mathbf{R}^{sub}$), TransFarToFar($\mathbf{R}^{sub,R}, \mathbf{R}^{sub}$)

the moments of the child nodes for the internal node case.

Determining the prunability of the given query and reference pair (shown in Algorithm 3.1.9). The function Summarize includes calls to the followings:

1. EvalFarFieldExpansion: evaluates the far-field moments stored in $\mathbf{R}^{sub}$ at each query point in $\mathbf{Q}^{sub}$ up to $(p_F + 1)^D$ terms. See Algorithm A.0.8.

2. AccumulateDirectLocalMoment: computes direct local moment contribution of R centered at $\mathbf{c}_{\mathbf{Q}^{sub}}$ in $\mathbf{Q}^{sub}$. See Algorithm A.0.10.

3. TransFarToLocal: translates the far-field moments of $\mathbf{R}^{sub}$ up to $(p_T + 1)^D$ terms to the local moment centered $\mathbf{c}_{\mathbf{Q}^{sub}}$ in $\mathbf{Q}^{sub}$. See Algorithm A.0.9.

Algorithm 3.1.8 CanSummarize($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \epsilon$): Determines the prunability of the given query node $\mathbf{Q}^{sub}$ and reference node $\mathbf{R}^{sub}$

return ChooseBestMethod $\left(\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \frac{\epsilon|\mathbf{R}^{sub}|\Phi^{l,new}(\mathbf{Q}^{sub} \times \mathbf{R})}{|\mathbf{R}|}\right) \neq \mathsf{D}$

Dual-tree Recursion. Algorithm 3.1.10 shows the basic structure of the dual-tree based KDE computation. This procedure is first called with $\mathbf{Q}$ and $\mathbf{R}$ as the root nodes of the query and the reference tree respectively. CanSummarize takes three

Algorithm 3.1.9 SUMMARIZE($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$): Summarizes the contribution of $\mathbf{R}^{sub}$.

{Add bound changes.}
$\Delta^l(\mathbf{Q}^{sub}) \leftarrow \Delta^l(\mathbf{Q}^{sub}) + \delta^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$, $\Delta^u(\mathbf{Q}^{sub}) \leftarrow \Delta^u(\mathbf{Q}^{sub}) + \delta^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$
if A is of the form $\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p_F)$ **then**
  EVALFARFIELDEXPANSION($\mathbf{R}^{sub}, \mathbf{Q}^{sub}, p_F$)
else if A is of the form $\mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}}, p_D)$ **then**
  ACCUMULATEDIRECTLOCALMOMENT($\mathbf{R}^{sub}, \mathbf{Q}^{sub}, p_D$)
else
  TRANSFARTOLOCAL($\mathbf{R}^{sub}, \mathbf{Q}^{sub}, p_T$)

parameters: the current query node $\mathbf{Q}^{sub}$, the current reference node $\mathbf{R}^{sub}$, and the global relative error tolerance ϵ. This function tests whether the the contribution of the given reference node for each query point in the given query node can be approximated within the error tolerance. If the approximation is not possible, then the algorithm continues to consider the query and the reference data at a finer granularity. The basic idea is to terminate the recursion as soon as possible by considering large "chunks" of the query data and the reference data and avoiding the number of exhaustive leaf-leaf computations. We can achieve this if we utilize approximation schemes that yield high accuracy and have cheap computational costs.

Each prune made for a pair of a query and a reference node is summarized in the given query node by incorporating the lower and the upper bound changes $\delta^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$ and $\delta^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$ contributed by the reference node into $\Delta^l(\mathbf{Q}^{sub})$ and $\Delta^u(\mathbf{Q}^{sub})$. These two bound updates due to a prune can be regarded as a new piece of information which is known only locally to the given query node $\mathbf{Q}^{sub}$. All of the bounds in the entire subtree of $\mathbf{Q}^{sub}$ should reflect this information. One way to achieve this effect is to pass the lower bound and the upper bound changes owned by $\mathbf{Q}^{sub}$ (i.e., $\Delta^l(\mathbf{Q}^{sub})$ and $\Delta^u(\mathbf{Q}^{sub})$) to $\mathbf{Q}^{sub}$'s immediate children, whenever the algorithm needs to consider the query dataset at a finer granularity by recursing to the left and the right child of $\mathbf{Q}^{sub}$. See Figure 16.

Algorithm 3.1.10 DFGT($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$): The core dual-tree routine for computing KDE.

$\delta^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub}) = |\mathbf{R}^{sub}|k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub}))$
$\delta^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub}) = |\mathbf{R}^{sub}|(k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - 1)$
{Add postponed contributions/bound changes from the current pair.}
$\Phi^{l,new}(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \Phi^{l,new}(\mathbf{Q}^{sub} \times \mathbf{R}) + \Delta^l(\mathbf{Q}^{sub}) + \delta^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$
$\Phi^{u,new}(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \Phi^{u,new}(\mathbf{Q}^{sub} \times \mathbf{R}) + \Delta^u(\mathbf{Q}^{sub}) + \delta^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$
if CanSummarize($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \epsilon$) **then**
  Summarize($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$)
else
  if $\mathbf{Q}^{sub}$ is a leaf node **then**
    if $\mathbf{R}^{sub}$ is a leaf node **then**
      DFGTBase($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$)
    else
      DFGT($\mathbf{Q}^{sub}, \mathbf{R}^{sub,L}$),DFGT($\mathbf{Q}^{sub}, \mathbf{R}^{sub,R}$)
  else
    {Push down postponed bound changes owned by $\mathbf{Q}^{sub}$ to the children.}
    $\Delta^l(\mathbf{Q}^{sub,L}) \leftarrow \Delta^l(\mathbf{Q}^{sub,L}) + \Delta^l(\mathbf{Q}^{sub})$, $\Delta^l(\mathbf{Q}^{sub,R}) \leftarrow \Delta^l(\mathbf{Q}^{sub,R}) + \Delta^l(\mathbf{Q}^{sub})$
    $\Delta^u(\mathbf{Q}^{sub,L}) \leftarrow \Delta^u(\mathbf{Q}^{sub,L}) + \Delta^u(\mathbf{Q}^{sub})$, $\Delta^u(\mathbf{Q}^{sub,R}) \leftarrow \Delta^u(\mathbf{Q}^{sub,R}) + \Delta^u(\mathbf{Q}^{sub})$
    $\Delta^l(\mathbf{Q}^{sub}) \leftarrow 0, \quad \Delta^u(\mathbf{Q}^{sub}) \leftarrow 0$
    if $\mathbf{R}^{sub}$ is a leaf node **then**
      DFGT($\mathbf{Q}^{sub,L}, \mathbf{R}^{sub}$), DFGT($\mathbf{Q}^{sub,R}, \mathbf{R}^{sub}$)
    else
      DFGT($\mathbf{Q}^{sub,L}, \mathbf{R}^{sub,L}$), DFGT($\mathbf{Q}^{sub,L}, \mathbf{R}^{sub,R}$)
      DFGT($\mathbf{Q}^{sub,R}, \mathbf{R}^{sub,L}$), DFGT($\mathbf{Q}^{sub,R}, \mathbf{R}^{sub,R}$)
    {Refine the bounds based on the recursion results.}
    $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \min\{\Phi^l(\mathbf{Q}^{sub,L} \times \mathbf{R}) + \Delta^l(\mathbf{Q}^{sub,L}), \Phi^l(\mathbf{Q}^{sub,R} \times \mathbf{R}) + \Delta^l(\mathbf{Q}^{sub,R})\}$
    $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \max\{\Phi^u(\mathbf{Q}^{sub,L} \times \mathbf{R}) + \Delta^u(\mathbf{Q}^{sub,L}), \Phi^u(\mathbf{Q}^{sub,R} \times \mathbf{R}) + \Delta^u(\mathbf{Q}^{sub,R})\}$

Base-case Computation. If a given leaf query and leaf reference node pair could not be pruned, then DFGTBase (shown in Algorithm 3.1.11) is called. Because all kernel evaluations are computed exactly, we can refine the bound summary statistics of the given query node Q (that is, $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})$ and $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R})$) further and hence we reset them to ∞ and $-\infty$ respectively. For each query point $\mathbf{q}_{\mathbf{i_m}} \in \mathbf{Q}^{sub}$, we first incorporate the postponed bound changes passed down from the ancestor node of $\mathbf{Q}^{sub}$. We loop over each reference point $\mathbf{r}_{\mathbf{j_n}} \in \mathbf{R}^{sub}$ and compute the kernel value between $\mathbf{q}_{\mathbf{i_m}}$ and $\mathbf{r}_{\mathbf{j_n}}$ and accumulate the lower bound $\Phi^l(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R})$, the kernel

Algorithm 3.1.11 DFGTBase($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$): Computes exact contribution of $\mathbf{R}^{sub}$ to $\mathbf{Q}^{sub}$.

$\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \infty, \qquad \Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow -\infty$
for each $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$ **do**
  {Add postponed changes passed down from the ancestor node of Q.}
  $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R}) \leftarrow \Phi^l(\mathbf{q_{i_m}}; \mathbf{R}) + \Delta^l(\mathbf{Q}^{sub}), \qquad \Phi^u(\mathbf{q_{i_m}}; \mathbf{R}) \leftarrow \Phi^u(\mathbf{q_{i_m}}; \mathbf{R}) + \Delta^u(\mathbf{Q}^{sub})$
  for each $\mathbf{r_{j_n}} \in \mathbf{R}^{sub}$ **do**
    $v \leftarrow k(\|\mathbf{q_{i_m}} - \mathbf{r_{j_n}}\|), \qquad \Phi^l(\mathbf{q_{i_m}}; \mathbf{R}) \leftarrow \Phi^l(\mathbf{q_{i_m}}; \mathbf{R}) + v$
    $\widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{E}}(\mathbf{q_{i_m}})) \leftarrow \widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{E}}(\mathbf{q_{i_m}})) + v$
    $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R}) \leftarrow \Phi^u(\mathbf{q_{i_m}}; \mathbf{R}) + (v - 1)$
  {Refine the bound summary statistics owned by $\mathbf{Q}^{sub}$.}
  $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \min\{\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}), \Phi^l(\mathbf{q_{i_m}}; \mathbf{R})\}$
  $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \max\{\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}), \Phi^u(\mathbf{q_{i_m}}; \mathbf{R})\}$
$\Delta^l(\mathbf{Q}^{sub}) \leftarrow 0, \qquad \Delta^u(\mathbf{Q}^{sub}) \leftarrow 0$

sum computed exhaustively $\widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{E}}(\mathbf{q_{i_m}}))$, and the upper bound $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$

We subtract one for updating $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$ for correcting the prior assumption that $k(||\mathbf{q_{i_m}} - \mathbf{r_{j_n}}||) = 1$, while the lower bound $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$ and $\widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{E}}(\mathbf{q_{i_m}}))$ are incremented by $k(||\mathbf{q_{i_m}} - \mathbf{r_{j_n}}||)$. As the contribution of the reference node $\mathbf{R}^{sub}$ is added onto the query point $\mathbf{q_{i_m}}$'s sum, we can refine the bound summary statistics owned by $\mathbf{Q}^{sub}$ such that $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) = \min_{\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}} \Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$ and $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) = \max_{\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}} \Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$. Finally, we reset the postponed bound changes stored in $\mathbf{Q}^{sub}$ to zero.

Post-processing (shown in Algorithm 3.1.12). For the non-leaf case, the local-to-local translation operator (TransLocalToLocal) is called to re-center the local moments at the current level and passes them down to the child nodes. For the leaf-case, EvalLocalExpansion is called to convert local moments to a single scalar that represents the contribution to a given query point.

3.1.10 Basic Properties of DFGT Algorithms

Theorem 3.1.8. *Lower/upper bounds are maintained properly at all times for each* $\mathbf{q} \in \mathbf{Q}^{sub}$ *and each query node* $\mathbf{Q}^{sub}$ *during the function call* DFGTMain.

Algorithm 3.1.12 DFGTPost($\mathbf{Q}^{sub}$): The post-processing routine.

if $\mathbf{Q}^{sub}$ is a leaf node **then**
  $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \infty, \quad \Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow -\infty$
  for each $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$ **do**
    {Add bound changes for the query node at the given query point $\mathbf{q_{i_m}}$.}
    $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R}) \leftarrow \Phi^l(\mathbf{q_{i_m}}; \mathbf{R}) + \Delta^l(\mathbf{Q}^{sub}), \quad \Phi^u(\mathbf{q_{i_m}}; \mathbf{R}) \leftarrow \Phi^u(\mathbf{q_{i_m}}; \mathbf{R}) + \Delta^u(\mathbf{Q}^{sub})$
    {Refine summary statistics for lower and upper bounds.}
    $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \min\{\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}), \Phi^l(\mathbf{q_{i_m}}; \mathbf{R})\}$
    $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \min\{\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}), \Phi^u(\mathbf{q_{i_m}}; \mathbf{R})\}$
    {Compute the contributions from the accumulated local moments.}
    $\widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{F2L}}(\mathbf{q_{i_m}})) \leftarrow$ EvalLocalExpansion($\mathbf{Q}^{sub}$)
    {Sum the contribution from the local moments (direct or translated), the far-field evaluations, and exhaustive evaluations.}
    $\widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}) \leftarrow \widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{DL}}(\mathbf{q_{i_m}}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{q_{i_m}})) + \widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{DF}}(\mathbf{q_{i_m}})) + \widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{E}})$
  $\Delta^l(\mathbf{Q}^{sub}) \leftarrow 0, \quad \Delta^u(\mathbf{Q}^{sub}) \leftarrow 0, \quad \widetilde{N}(\mathbf{Q}^{sub}) \leftarrow 0$
else
  TransLocalToLocal($\mathbf{Q}^{sub}, \mathbf{Q}^{sub,L}$), TransLocalToLocal($\mathbf{Q}^{sub}, \mathbf{Q}^{sub,R}$)
  $\Delta^l(\mathbf{Q}^{sub,L}) \leftarrow \Delta^l(\mathbf{Q}^{sub,L}) + \Delta^l(\mathbf{Q}^{sub})$, $\Delta^l(\mathbf{Q}^{sub,R}) \leftarrow \Delta^l(\mathbf{Q}^{sub,R}) + \Delta^l(\mathbf{Q}^{sub})$
  $\Delta^u(\mathbf{Q}^{sub,L}) \leftarrow \Delta^u(\mathbf{Q}^{sub,L}) + \Delta^u(\mathbf{Q}^{sub})$, $\Delta^u(\mathbf{Q}^{sub,R}) \leftarrow \Delta^u(\mathbf{Q}^{sub,R}) + \Delta^u(\mathbf{Q}^{sub})$
  $\widetilde{N}(\mathbf{Q}^{sub}) \leftarrow 0$, $\Delta^l(\mathbf{Q}^{sub}) \leftarrow 0$, $\Delta^u(\mathbf{Q}^{sub}) \leftarrow 0$
  DFGTPost(Q^L), DFGTPost(Q^R)
  {Refine the bounds based on the results of the recursion.}
  $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \min\{\Phi^l(\mathbf{Q}^{sub,L} \times \mathbf{R}), \Phi^l(\mathbf{Q}^{sub,R} \times \mathbf{R})\}$
  $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \max\{\Phi^u(\mathbf{Q}^{sub,L} \times \mathbf{R}), \Phi^u(\mathbf{Q}^{sub,R} \times \mathbf{R})\}$

Proof. We show that the bounds are maintained properly for three main parts in the function DFGTMain: DFGTInitQ, DFGT, and DFGTPost.

The function call DFGTInitQ: It is clear that for all $\mathbf{q_i} \in \mathbf{Q}$, $0 = \Phi^l(\mathbf{q_i}; \mathbf{R}) \leq \Phi(\mathbf{q_i}; \mathbf{R}) \leq \Phi^u(\mathbf{q_i}; \mathbf{R}) = |\mathbf{R}|$. Furthermore, for each query node Q, $0 = \Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leq \Phi(\mathbf{q_{i_m}}; \mathbf{R}) \leq \Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) = |\mathbf{R}|$ for each $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$.

The function call DFGTBase: Let $\mathbf{Q}^{sub}$ and $\mathbf{R}^{sub}$ be the query node and the reference node respectively. For each query point $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$, $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$ is incremented by $\Delta^l(\mathbf{Q}^{sub}) + \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} k(||\mathbf{q_{i_m}} - \mathbf{r_{j_n}}||)$, and $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$ by $\Delta^u(\mathbf{Q}^{sub}) + \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} (k(||\mathbf{q_{i_m}} - \mathbf{r_{j_n}}||) - 1)$; this operation incorporates the passed-down contribution for $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$, and un-does the assumption made during the initialization phase

of DFGTInitQ. $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})$ and $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R})$ are updated to be the minimum among $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$ and the maximum among $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$ respectively. The postponed bound changes $\Delta^l(\mathbf{Q}^{sub})$ and $\Delta^u(\mathbf{Q}^{sub})$ are cleared to avoid double-counting when $\mathbf{Q}^{sub}$ may be visited later.

The function call DFGT: We induct on the number of points owned by the query node $\mathbf{Q}^{sub}$ and the reference node $\mathbf{R}^{sub}$ in consideration (i.e. $|\mathbf{Q}^{sub}|+|\mathbf{R}^{sub}|$). The only possible places that change $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$, $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$, $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})$ and $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R})$ are the call to the base case function DFGTBase and the last two lines of the function DFGT. The correctness of DFGTBase function is proven already, so we consider the second case. The two function calls DFGT$(\mathbf{Q}^{sub,L}, \mathbf{R}^{sub})$ and DFGT$(\mathbf{Q}^{sub,R}, \mathbf{R}^{sub})$ (in case R is a leaf node) and the four function calls DFGT$(\mathbf{Q}^{sub,L}, \mathbf{R}^{sub,L})$, DFGT$(\mathbf{Q}^{sub,L}, \mathbf{R}^{sub,R})$, DFGT$(\mathbf{Q}^{sub,R}, \mathbf{R}^{sub,L})$, and DFGT$(\mathbf{Q}^{sub,R}, \mathbf{R}^{sub,R})$ (in case $\mathbf{R}^{sub}$ is an internal node) are smaller subproblems than $(\mathbf{Q}^{sub}, (\mathbf{R}^{sub})$ pair. By the induction hypothesis, these calls maintain the lower and the upper bounds properly. The lower bound is set to the minimum of the "best" lower bound owned by the children of $\mathbf{Q}^{sub}$: $\min\{\Phi^l(\mathbf{Q}^{sub,L} \times \mathbf{R}) + \Delta^l(\mathbf{Q}^{sub,L}), \Phi^l(\mathbf{Q}^{sub,R} \times \mathbf{R}) + \Delta^l(\mathbf{Q}^{sub,R})\}$. Similarly, the upper bound is set to the maximum of the "best" upper bound owned by the children of $\mathbf{Q}^{sub}$: $\max\{\Phi^u(\mathbf{Q}^{sub,L} \times \mathbf{R}) + \Delta^u(\mathbf{Q}^{sub,L}), \Phi^u(\mathbf{Q}^{sub,R} \times \mathbf{R}) + \Delta^u(\mathbf{Q}^{sub,R})\}$.

The function call DFGTPost: We again induct on the number of points owned by the query node $\mathbf{Q}^{sub}$ passed in as the argument to this function. If the query node $\mathbf{Q}^{sub}$ is a leaf node, each query point $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$ incorporates the passed-down bound changes $\Delta^l(\mathbf{Q}^{sub})$ and $\Delta^u(\mathbf{Q}^{sub})$. The bounds $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})$ and $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R})$ are (correctly) set to the minimum among $\Phi^l(\mathbf{q_{i_m}}; \mathbf{R})$ and the maximum among $\Phi^u(\mathbf{q_{i_m}}; \mathbf{R})$. If $\mathbf{Q}^{sub}$ is not a leaf node: we know the sub-calls DFGTPost$(\mathbf{Q}^{sub,L})$ and DFGTPost$(\mathbf{Q}^{sub,R})$ maintains correct lower and upper bounds by the induction

hypothesis since $\mathbf{Q}^{sub,L}$ and $\mathbf{Q}^{sub,R}$ contain a smaller number of points. Setting the lower and upper bounds for $\mathbf{Q}^{sub}$ by the operations: $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \min\{\Phi^l(\mathbf{Q}^{sub,L} \times \mathbf{R}), \Phi^l(\mathbf{Q}^{sub,R} \times \mathbf{R})\}$, $\Phi^u(\mathbf{Q}^{sub} \times \mathbf{R}) \leftarrow \max\{\Phi^u(\mathbf{Q}^{sub,L} \times \mathbf{R}), \Phi^u(\mathbf{Q}^{sub,R} \times \mathbf{R})\}$ is valid. □

Theorem 3.1.9. *After calling* DFGTPost *(Algorithm 3.1.12) in* DFGTMain *(Algorithm 3.1.5), each query point* $\mathbf{q_i} \in \mathbf{Q}$ *accounts for every reference point* $r_j \in \mathbf{R}$ *in its Gaussian kernel sum approximation* $\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R})$.

Proof. In Algorithm 3.1.10, for each $\mathbf{q_i} \in \mathbf{Q}$, each $\mathbf{r_j} \in \mathbf{R}$ is either accounted by an exhaustive computation in DFGTBase or a prune in Summarize. All exhaustive computations for $\mathbf{q_i} \in \mathbf{Q}$ directly update $\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R}^{\mathsf{E}}(\mathbf{q_i}))$, while any pruned contributions will be incorporated into each $\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R}^{\mathsf{F2L}}(\mathbf{q_i}))$ (hence into $\widetilde{\Phi}(\mathbf{q_i}, \mathbf{R}(\mathbf{q_i}))$) and when they are pushed down (to the leaf node to which $\mathbf{q_i}$ belongs) during the DFGT recursion or DFGTPost. □

Theorem 3.1.10. *For each query point* $\mathbf{q_i} \in \mathbf{Q}$*, the approximated kernel sum* $\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R})$ *satisfies the global relative error tolerance* ϵ.

Proof. For simplicity, let us limit the available approximation methods to $\mathsf{A} \in \{\mathsf{D}, \widetilde{\mathsf{N}}(\mathbf{c}, 0)\}$ where D denotes the exhaustive computation and $\widetilde{\mathsf{N}}(\mathbf{c}, 0)$ denotes the centroid-based approximation about $\mathbf{c}$.

Given $\mathbf{q_i} \in \mathbf{Q}$, let $\mathbf{Q'}$ be the (unique) leaf node that owns $\mathbf{q_i}$. Let $\mathbf{R}^{\mathsf{F2L}}(\mathbf{q_i}) = \{\mathbf{R}^{T_a}\}_{a=1}^{N_a}$ denote the set of reference nodes whose kernel sum contribution were accounted via centroid approximation and $\mathbf{R}^{\mathsf{E}}(\mathbf{q_i}) = \{R_{E_b}\}_{b=1}^{N_b}$ the set of reference nodes whose kernel sum contribution were computed exhaustively. Then it is clear that $\mathbf{R} = \left(\bigcup_{a=1}^{N_a} \mathbf{R}^{T_a}\right) \cup \left(\bigcup_{b=1}^{N_b} \mathbf{R}^{E_b}\right)$ with $\mathbf{R}^{T_{a'}} \cap \mathbf{R}^{T_{a''}} = \emptyset$, $\mathbf{R}^{E_{b'}} \cap \mathbf{R}^{E_{b''}} = \emptyset$, $\mathbf{R}^{T_{a'}} \cap \mathbf{R}^{E_{b'}} = \emptyset$ for $1 \leq a', a'' \leq N_a$ and $1 \leq b', b'' \leq N_b$. Let $\mathbf{Q}^{T_a}$ be the query node that owns $\mathbf{q_i}$ and is considered with the reference node $\mathbf{R}^{T_a}$ and pruned. Let $\Phi^{l(a)}(\mathbf{Q}^{T_a} \times \mathbf{R})$ be a "snapshot" of the running lower bound on the kernel sum for query points owned

by $\mathbf{Q}^{T_a}$ at the time the query node $\mathbf{Q}^{T_a}$ and the reference node $\mathbf{R}^{T_a}$ were considered (and subsequently pruned). By the triangle inequality:

$$
\begin{aligned}
&\left|\widetilde{\Phi}(\mathbf{q_i};\mathbf{R}) - \Phi(\mathbf{q_i};\mathbf{R})\right| \\
=&\left|\widetilde{\Phi}\left(\mathbf{q_i};\left(\bigcup_{a=1}^{N_a}\left\{(\mathbf{R}^{T_a},\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}},0))\right\}\right)\cup\left(\bigcup_{b=1}^{N_b}\left\{(\mathbf{R}^{E_b},\mathsf{D})\right\}\right)\right) - \Phi\left(\mathbf{q_i};\left(\bigcup_{a=1}^{N_a}\mathbf{R}^{T_a}\right)\cup\left(\bigcup_{b=1}^{N_b}\mathbf{R}^{E_b}\right)\right)\right| \\
\leq&\left|\left(\sum_{a=1}^{N_a}\widetilde{\Phi}\left(\mathbf{q_i};\{(\mathbf{R}^{T_a},\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}},0))\}\right) - \Phi(\mathbf{q_i};\mathbf{R}^{T_a})\right) + \left(\sum_{b=1}^{N_b}\widetilde{\Phi}\left(\mathbf{q_i},\{(\mathbf{R}^{E_b},\mathsf{D})\}\right) - \Phi(\mathbf{q_i},\mathbf{R}^{E_b})\right)\right| \\
\leq&\sum_{a=1}^{N_a}\left|\widetilde{\Phi}\left(\mathbf{q_i};\{(\mathbf{R}^{T_a},\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}},0))\}\right) - \Phi(\mathbf{q_i};\mathbf{R}^{T_a})\right| + \sum_{b=1}^{N_b}\left|\widetilde{\Phi}\left(\mathbf{q_i};\{(\mathbf{R}^{E_b},\mathsf{D})\}\right) - \Phi(\mathbf{q_i};\mathbf{R}^{E_b})\right| \\
\leq&\sum_{a=1}^{N_a}|\mathbf{R}^{T_a}|\max\left\{\begin{array}{c}\left|k(d^u(\mathbf{Q}^{T_a},\mathbf{R}^{T_a})) - k(||\mathbf{c}_{\mathbf{Q}^{T_a}} - \mathbf{c}_{\mathbf{R}^{T_a}}||)\right|, \\ \left|k(d^l(\mathbf{Q}^{T_a},\mathbf{R}^{T_a})) - k(||\mathbf{c}_{\mathbf{Q}^{T_a}} - \mathbf{c}_{\mathbf{R}^{T_a}}||)\right|\end{array}\right\} + \sum_{b=1}^{N_b}|\mathbf{R}^{E_b}|\cdot 0 \\
\leq&\sum_{a=1}^{N_a}\frac{|\mathbf{R}^{T_a}|\epsilon}{|\mathbf{R}|}\Phi^{l(a)}(\mathbf{Q}^{T_a}\times\mathbf{R}) + \sum_{b=1}^{N_b}\frac{|\mathbf{R}^{E_b}|\epsilon}{|\mathbf{R}|}\Phi^{l(b)}(\mathbf{Q}'\times\mathbf{R}) \\
\leq&\sum_{a=1}^{N_a}\frac{|\mathbf{R}^{T_a}|\epsilon}{|\mathbf{R}|}\Phi(\mathbf{q_i};\mathbf{R}) + \sum_{b=1}^{N_b}\frac{|\mathbf{R}^{E_b}|\epsilon}{|\mathbf{R}|}\Phi(\mathbf{q_i};\mathbf{R}) \leq \epsilon\Phi(\mathbf{q_i};\mathbf{R})
\end{aligned}
$$

The proof can be easily extended to the case with four available approximation methods $\mathsf{A}\in\{\mathsf{D},\widetilde{\mathsf{N}}(\mathbf{c},p),\mathsf{F}(\mathbf{c},p),\mathsf{N}(\mathbf{c},p)\}$ □

3.2 Experimental Results

We evaluated empirical performance of six algorithms:

- Naive: the brute-force algorithm (Algorithm 3.3.1).
- FFT: Fast fourier transform based kernel density estimate [192].
- FGT: Fast Gauss transform [84].
- IFGT: improved fast Gauss transform [201, 154].
- DFD: the dual-tree centroid-based approximation method [80, 82].
- DFGT: our new algorithm (Algorithm 3.1.5).

We used the following six real-world datasets:

Table 1: Empirical comparison of six different algorithms on different magnitudes of bandwidths on three different datasets. Each entry in the table has a timing number (if finite), ∞ symbol (if no parameter tweaking could achieve the error tolerance), X symbol (if the algorithm segfaulted).

Alg\Scale	*0.001*	*0.01*	*0.1*	*1*	*10*	*100*	*1000*	Σ
sj2-50000-2, $D = 2, N = 50000, h^*_{CV_{LS}} = 0.00139506$								
Naive	241	241	241	241	241	241	241	1687
FFT	∞	∞	∞	∞	∞	1.02	0.03	∞
FGT	X	X	X	2.63	1.48	0.33	0.18	X
IFGT	∞	∞	∞	155	7.26	0.40	0.03	∞
DFD	1.58	1.63	2.14	4.33	39.7	29.5	1.51	80.39
DFGT	0.43	0.47	1.00	3.48	21	2.48	0.96	29.8
colors50k, $D = 2, N = 50000, h^*_{CV_{LS}} = 0.0016911$								
Naive	241	241	241	241	241	241	241	1687
FFT	∞	∞	∞	∞	∞	∞	0.16	∞
FGT	X	X	X	120	10	4	0.22	X
IFGT	∞	∞	∞	∞	∞	0.54	0.07	∞
DFD	1.62	1.76	2.36	12.5	102	17.0	2.41	139.65
DFGT	0.44	0.60	1.21	15.6	20	4.20	0.67	42.7
bio5, $D = 5, N = 103010, h^*_{CV_{LS}} = 0.000308646$								
Naive	1310	1310	1310	1310	1310	1310	1310	9170
FFT	X	X	X	X	X	X	X	X
FGT	X	X	X	X	X	X	X	X
IFGT	∞	∞	∞	∞	∞	∞	1.04	∞
DFD	0.34	0.36	0.92	6.31	113	643	125	888.93
DFGT	0.35	0.37	0.94	6.51	102	304	121	535.17

- *sj2-50000-2*: two-dimensional astronomy position dataset.
- *colors50k*: two-dimensional astronomy color dataset.
- *bio5*: five-dimensional pharmaceutical dataset.
- *edgsc-radec*: two-dimensional astronomy angle dataset.
- *mockgalaxy-D-1M*: three-dimensional astronomy position dataset.
- *psf1-psf4-stargal-2d-only*: two-dimensional astronomy dataset.

Note that the last three datasets contain over 1 million points and demonstrate the scalability of our fast algorithm. For each dataset, we evaluated the empirical performance on computing kernel density estimates at seven different bandwidths ranging

Table 2: Empirical comparison of three algorithms on different magnitudes of bandwidths on three larger datasets. All timings are reported in seconds.

Alg\Scale	*0.001*	*0.01*	*0.1*	*1*	*10*	*100*	*1000*	Σ
edsgc-radec, $D = 2, N = 1495877, h^*_{CV_{LS}} = 0.000473061$								
Naive	2.2e5	2.2e5	2.2e5	2.2e5	2.2e5	2.2e5	2.2e5	1.5e6
DFD	4.9e1	4.9e1	6.3e1	1e2	1.5e3	2e4	1.3e3	2.3e4
DFGT	6.8e0	7.4e0	2.1e1	5.9e1	1.7e3	3.5e3	1.4e2	5.4e3
mockgalaxy-D-1M, $D = 3, N = 1000000, h^*_{CV_{LS}} = 0.00010681$								
Naive	9.6e4	9.6e4	9.6e4	9.6e4	9.6e4	9.6e4	9.6e4	6.7e5
DFD	2.4e0	2.4e0	2.6e0	1.5e1	9.7e1	1.7e2	4.4e3	4.7e3
DFGT	2.4e0	2.4e0	2.6e0	1.5e1	1.1e2	2.1e2	4e3	4.3e3
psf1-psf4-stargal-2d-only, $D = 2, N = 3056092, h^*_{CV_{LS}} = 0.00489463$								
Naive	9e5	9e5	9e5	9e5	9e5	9e5	9e5	6.3e6
DFD	1.1e2	1.5e2	1.2e3	2.2e4	3.9e4	2.9e3	1.1e2	6.5e4
DFGT	3.9e1	8.1e1	1.4e3	1.6e4	2.3e3	1.9e2	4.2e1	1.9e4

from 10^{-3} to 10^3 times the optimal bandwidths according to the standard least-squares cross-validation score [170]. We measured the time required for computing KDE estimates that guarantee the global relative error: $\left|\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R}) - \Phi(\mathbf{q_i}; \mathbf{R})\right| \leq \epsilon\Phi(\mathbf{q_i}; \mathbf{R})$. We used $\epsilon = 0.01$. Each entry in the table has a timing number (if finite), ∞ symbol (if no parameter tweaking could achieve the error tolerance), X symbol (if the algorithm segfaulted; this is common in grid-based algorithms in higher dimension). The entries under Σ symbol denote the total time for least-squares cross-validation. Note that the FGT ensures: $\left|\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R}) - G(\mathbf{q_i}; \mathbf{R})\right| \leq \tau$. Therefore, we first set $\tau = \epsilon$, halving τ until the error tolerance ϵ was met; the time for verifying the global error guarantee (which includes comparison against the naively computed results) was not included in the timing. For the FFT, we started with 16 grid points along each dimension, and doubled the number of grid points until the error guarantee was met. For the IFGT, we took the most recent version of the algorithm that does automatic parameter tuning described in [154]. Our algorithms based on dual-tree methods guarantees the error bound automatically via a direct parameter ϵ.

The naive timings for the last datasets have been extrapolated from the performances on the smaller datasets. Our results demonstrate that our new algorithm can

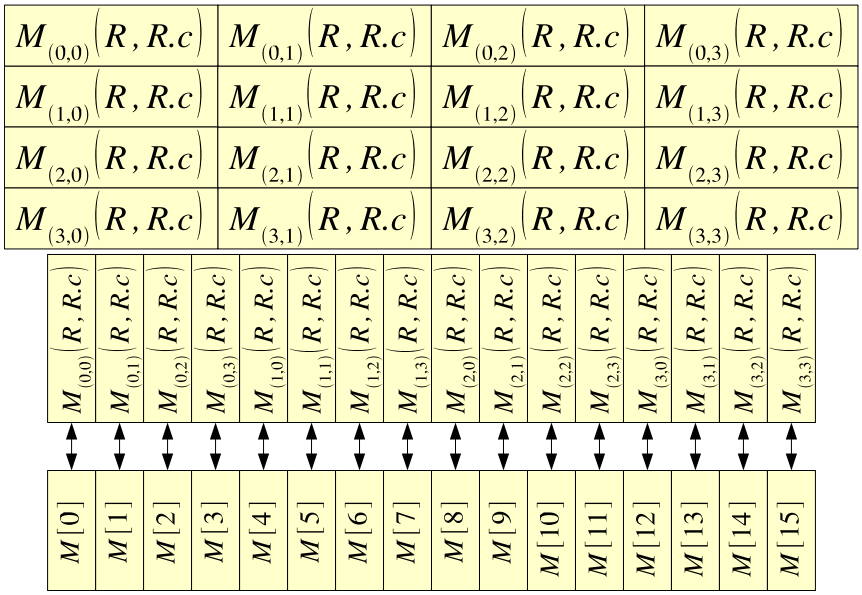


Figure 26: Top: It is conceptually easy to visualize the moments to be stored in a multi-dimensional array. Each dimension iterates over $(p_{max}+1)$ scalars, a total count of $(p_{max}+1)^D$ scalars. **Bottom:** The linear layout for the storing the coefficients.

be as 15 times as fast as the original dual-tree algorithm. As expected, the grid-based original fast Gauss transform and the fast Fourier transformed based method fails in dimensions above two.

3.3 Applications in Nonparametric Density Estimation

Kernel density estimation (KDE) is the most widely used and studied nonparametric density estimation method. The model is the reference dataset $\mathbf{R}$ itself, containing the reference points indexed by natural numbers. Assume a local kernel function $k(\cdot)$ centered upon each reference point, and its scale parameter h (the 'bandwidth'). The common choices for $k(\cdot)$ include the spherical, Gaussian and Epanechnikov kernels. We are given the query dataset $\mathbf{Q}$ containing query points whose densities we want to predict. The density estimate at the i-th query point $\mathbf{q_i} \in \mathbf{Q}$ is:

$$\hat{p}_h(\mathbf{q_i}) = \frac{1}{|\mathbf{R}|} \sum_{\mathbf{r_j} \in \mathbf{R}} \frac{1}{V_{Dh}} k\left(||\mathbf{q_i} - \mathbf{r_j}||\right) \tag{3.3.1}$$

where $V_{Dh} = \int_{-\infty}^{\infty} k(z)dz$, a normalizing constant depending on D and h. With no assumptions on the true underlying distribution, if $h \to 0$ and $|\mathbf{R}|h \to \infty$ and $k(\cdot)$

Algorithm 3.3.1 NAIVEKDE($\mathbf{Q}, \mathbf{R}$): A brute-force computation of KDE.

```
for each q_i ∈ Q do
    Φ(q_i; R) ← 0
    for each r_j ∈ R do
        Φ(q_i; R) ← Φ(q_i; R) + k(||q_i − r_j||)
    Normalize each G(q_i, R)
```

satisfy some mild conditions:

$$\int |\hat{p}_h(\mathbf{x}) - p(\mathbf{x})| \mathbf{dx} \to 0 \tag{3.3.2}$$

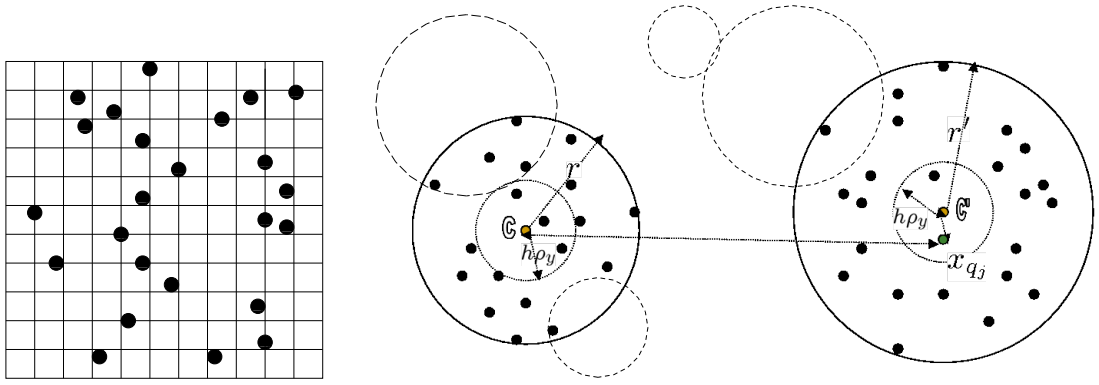
as $|\mathbf{R}| \to \infty$ with probability 1. As more data are observed, the estimate converges to the true density. In order to build our model for evaluating the densities at each $\mathbf{q_i} \in \mathbf{Q}$, we need to find the initially unknown asymptotically optimal bandwidth h^* for the given reference dataset $\mathbf{R}$. There are two main types of cross-validation methods for selecting the asymptotically optimal bandwidth. Cross-validation methods use the reference dataset $\mathbf{R}$ as the query dataset $\mathbf{Q}$ (i.e. $\mathbf{Q} = \mathbf{R}$). *Likelihood cross-validation* is derived by minimizing the Kullback-Leibler divergence $\int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{\hat{p}_h(\mathbf{x})} \mathbf{dx}$, which yields the score:

$$CV_{LK}(h) = \frac{1}{|\mathbf{R}|} \sum_{\mathbf{r_j} \in \mathbf{R}} \log \hat{p}_{h,-j}(\mathbf{r_j}) \tag{3.3.3}$$

where the $-j$ subscript denotes an estimate using all $|\mathbf{R}|$ points except the j-th reference point. The bandwidth $h^*_{CV_{LK}}$ that maximizes $CV_{LK}(h)$ is an asymptotically optimal bandwidth in likelihood cross validation sense. *Least-squares cross-validation* minimizes the integrated squared error
$\int \left(\hat{p}_h(\mathbf{x}) - p(\mathbf{x})\right)^2 \mathbf{dx}$, yielding the score:

$$CV_{LS}(h) = \frac{1}{|\mathbf{R}|} \sum_{r_j \in \mathbf{R}} \left(\hat{p}^*_{-j}(\mathbf{r_j}) - 2\hat{p}_{-j}(\mathbf{r_j})\right) \tag{3.3.4}$$

where $\hat{p}^*_{-j}(\cdot)$ is evaluated using the convolution kernel $k(\cdot) * k(\cdot)$. For the Gaussian kernel with bandwidth of h, the convolution kernel $k(\cdot) * k(\cdot)$ is the Gaussian kernel with bandwidth of $2h$. Both cross validation scores require $|\mathbf{R}|$ density estimate based on $|\mathbf{R}| - 1$ points, yielding a brute-force computational cost scaling **quadratically**



(a) (b)

Figure 27: (a) Grid structure used in fast Gauss transform and multidimensional fast Fourier transform. (b) Single-level Clustering structure used in improved fast Gauss transform.

(that is $O(|\mathbf{R}|^2)$) (see Algorithm 3.3.1). To make matters worse, nonparametric methods require a large number of reference points for convergence to the true underlying distribution and this has prevented many practitioners from applying nonparametric methods for function estimation.

3.3.1 Previous Approaches

There are three main approaches proposed for overcoming the computational barrier in evaluating the Gaussian kernel sums:

1. to expand the kernel sum as a power series [84, 201, 154] using a grid or a flat-clustering.

2. to express the kernel sum as a convolution sum by using the grid of field charges created from the dataset [192].

3. to utilize an adaptive hierarchical structure to group data points based on proximity [82, 77, 80].

Now we briefly describe the strengths and the weaknesses of these methods.

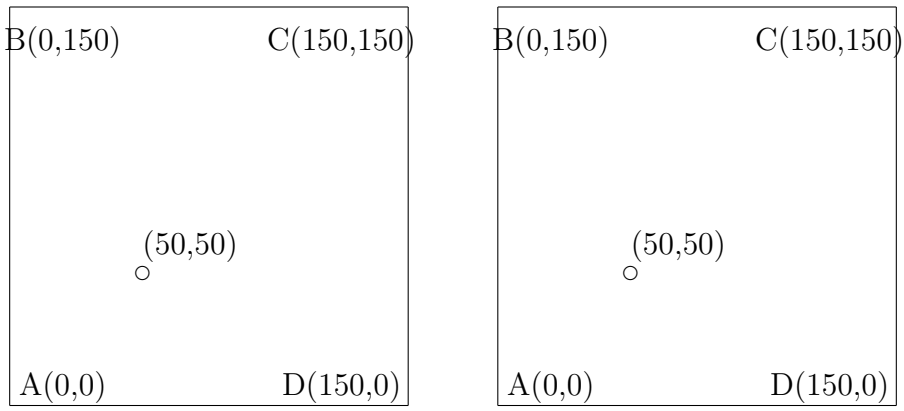
The Fast Gauss Transform (FGT). FGT [84] belongs to a family of methods

called the Fast Multipole Methods (FMM). These family of methods come with rigorous error bound on the kernel sums. Unlike other FMM algorithms, FGT uses a grid structure (see Figure 27(a)) whose maximum side length is restricted to be at most the bandwidth h used in cross-validation due to the error bound criterion. FGT has not been widely used in higher dimensional statistical contexts. First, the number of the terms in the power series expansion for the kernel sums grows exponentially with dimensionality D; this causes computational bottleneck in evaluating the series expansion or translating a series expansion from one center to another. Second, the grid structure is extremely inefficient in higher dimensions since the storage cost is exponential in D and many of the boxes will be empty.

The Improved Fast Gauss Transform (IFGT). IFGT is similar to FMM but utilizes a flat clustering to group data points (see Figure 27(b)), which is more efficient than a grid structure used in FGT. The number of clusters k is chosen in advance. A partition of the data points into $\mathbf{C_1}, \cdots, \mathbf{C_k}$ is formed so that each reference point $\mathbf{r_j} \in \mathbf{R}$ is grouped according to its proximity to the set of representative points $\mathbf{c_1}, \cdots, \mathbf{c_k}$. That is, $\mathbf{r_j} \in \mathbf{C_m}$ if and only if $||\mathbf{r_j} - \mathbf{c_m}|| \leq ||\mathbf{r_j} - \mathbf{c_l}||$ for $1 \leq l \leq k$.

Furthermore, IFGT proposes using a different series expansion that does not require translation of expansion centers as done in FGT. The original algorithm [201] required tweaking of multiple parameters which did not offer for a user to control the accuracy of the approximation. The latest version [154] is now fully automatic in choosing the approximation parameter for the absolute error bound, but is still inefficient except on large bandwidth parameters. See Section 3.2.

Fast Fourier Transform (FFT). FFT is often quoted as the solution to the computational problem in evaluating the Gaussian kernel sums. Gaussian kernel summation using FFT is described in [171] and [192]. [171] discusses the implementation of KDE only in a univariate case, while [192] extends [171] to handle more than one



(a) Nearest Neighbor Binning Rule ($A = 1, B = C = D = 0$)

(b) Linear Binning Rule ($A = \frac{4}{9}, B = \frac{2}{9}, C = \frac{1}{9}, D = \frac{2}{9}$)

Figure 28: Two possible binning rules for KDE using multidimensional fast Fourier transform. Consider a data point falling in a two-dimensional rectangle. In 28(a), the entire weight is assigned to the nearest grid point. In 28(b), the weight is distributed to all neighboring grid points by linear interpolation.

dimension. It uses a grid structure shown in Figure 27(a) by specifying the number of grid points along each dimension.

The algorithm first computes the $M_1 \times \cdots \times M_D$ matrix by binning the data assigning the raw data to neighboring grid points using one of the binning rules. This involves computing the minimum and maximum coordinate values $(g_{i,M_i}, g_{i,1})$, and the grid width $\delta_i = \frac{g_{i,M_i} - g_{i,1}}{M_i - 1}$ for each i-th dimension. This essentially divides each i-th dimension into $(M_i - 1)$ intervals of equal length. In particular, [192] discusses two different types of binning rules - linear binning, which is recommended by Silverman, and nearest-neighbor binning. [192] states that nearest-neighbor binning rule performs poorly, so we will test the implementation using the linear binning rule, as recommended by both authors. In addition, we compute the $L_1 \times \cdots \times L_D$ kernel weight matrix, where $L_i = \min\left(\left\lfloor \frac{\tau h}{\delta_i} \right\rfloor, M_i - 1\right)$, with $\tau \approx 4$ and $K_l = \prod_{k=1}^{d} \exp\left(\frac{-0.5 l_k \delta_k}{h^2}\right)$, $-L_k \leq l_k \leq L_k$, for $l = [l_1, ..., l_D]^T \in \mathbb{Z}^D$.

To reduce the wrap-around effects of fast Fourier transform near the dataset boundary, we appropriately zero-pad the grid count and the kernel weight matrices to two matrices of the dimensionality $P_1 \times \cdots P_D$, where $P_i = 2^{\log_2 \lceil M_i + L_i \rceil}$. The

$$\text{The grid count matrix: } \mathbf{C}^Z = \begin{pmatrix} c_{1,1} & \cdots & c_{1,M_2} & \\ \vdots & \ddots & \vdots & \mathbf{0} \\ c_{M_1,1} & \cdots & c_{M_1,M_2} & \\ & \mathbf{0} & & \mathbf{0} \end{pmatrix}$$

$$\text{The kernel weight matrix: } \mathbf{K}^Z = \begin{pmatrix} K_{00} & \cdots & K_{0L_2} & & K_{0L_2} & \cdots & K_{01} \\ \vdots & \ddots & \vdots & \mathbf{0} & \vdots & \ddots & \vdots \\ K_{L_10} & \cdots & K_{L_1L_2} & & K_{L_1L_2} & \cdots & K_{L_11} \\ & \mathbf{0} & & \mathbf{0} & & \mathbf{0} & \\ K_{L_10} & \cdots & K_{L_1L_2} & & K_{L_1L_2} & \cdots & K_{L_11} \\ \vdots & \ddots & \vdots & \mathbf{0} & \vdots & \ddots & \vdots \\ K_{10} & \cdots & K_{1L_2} & & K_{1L_2} & \cdots & K_{11} \end{pmatrix}$$

$$\text{where } K_{l_1,l_2} = e^{\frac{-0.5((l_1\delta_1)^2+(l_2\delta_2)^2)}{h^2}}.$$

Figure 29: The grid count and the kernel weight matrix formed for a two-dimensional dataset. They are formed by appropriately zero-padding for taking the boundary-effects of fast Fourier transform based algorithms into account.

key ingredient in this method is the use of Convolution Theorem for Fourier transforms. The structure of the computed grid count matrix and the kernel weight matrix is crafted to take advantage of the fast Fourier transform. For every grid point $\mathbf{g_j}$, $\tilde{s_k}(\mathbf{g_j}) = \sum_{l_1=-L_1}^{L_1} \cdots \sum_{l_D=-L_D}^{L_D} c_{j-l} K_{k,l}$ can be computed using the Convolution Theorem for Fourier Transform. After taking the convolution of the grid count matrix and the kernel weight matrix, the $M_1 \times \cdots \times M_D$ sub-matrix in the upper left corner of the resultant matrix contains the kernel density estimate of the grid points. The density estimate of each query point is then linearly interpolated using the density estimates of neighboring grid points inside the cell it falls into. However, performing a calculation on equally-spaced grid points introduces artifacts at the boundaries of the data. The linear interpolation of the data points by assigning to neighboring grid points introduce further errors. Increasing the number of grid points to use along each dimension can provide more accuracy but also require more space to store the grid. Moreover, it is impossible to directly quantify incurred error on each estimate in terms of the number of grid points.

Dual-tree KDE. In terms of discrete algorithmic structure, the dual-tree framework of [78] generalizes all of the well-known kernel summation algorithms. These include the Barnes-Hut algorithm [10], the Fast Multipole Method [83], Appel's algorithm [7], and the WSPD [28]: the dual-tree method is a node-node algorithm (considers query regions rather than points), is fully recursive, can use distribution-sensitive data structures such as *kd*-trees, and is bichromatic (can specialize for differing query set $\mathbf{Q}$ and reference set $\mathbf{R}$). It was applied to the problem of kernel density estimation in [82] using a simple variant of a centroid approximation used in [7].

This algorithm is currently the fastest Gaussian kernel summation algorithm for general dimensions. Unfortunately, when performing cross-validation to determine the (initially unknown) optimal bandwidth, both sub-optimally small and large bandwidths must be evaluated. Section 3.2 demonstrates that the dual-tree method tends to be efficient at the optimal bandwidth and at bandwidths below the optimal bandwidth and at very large bandwidths. However, its performance degrades for intermediately large bandwidths.

3.3.2 Conclusion

In this chapter we presented an improvement to the dual-tree algorithm [82, 77, 80], the first practical kernel summation algorithm for general dimension. Our extension is based on the series-expansion for the Gaussian kernel used by fast Gauss transform [84]. First, we derive two additional analytical machinery for extending the original algorithm to utilize a adaptive hierarchical data structure called *kd*-trees [15], demonstrating the first truly hierarchical fast Gauss transform, which we call the Dual-tree Fast Gauss Transform (DFGT). Second, we show how to integrate the series-expansion approximation within the dual-tree approach to compute kernel summations with a user-controllable relative error bound. We evaluate our algorithm on real-world datasets in the context of optimal bandwidth selection in kernel density

estimation. Our results demonstrate that our new algorithm is the only one that guarantees a relative error bound and offers fast performance across a wide range of bandwidths evaluated in cross validation procedures. Our results demonstrate that the $\mathcal{O}(p^D)$ expansion helps reduce the computational time on datasets of dimensionality up to 5. We note that our method has been included in the thesis [197] for comparison.

CHAPTER IV

SERIES EXPANSION-BASED METHOD II

Here we again consider the acceleration of the Gaussian kernel sums (Equation (3.0.2)) with arbitrary non-negative weights w_j[1]. For concreteness, we again define the computational task tackled in this chapter.

Problem: Suppose we are given the set of query points $\mathbf{Q}$ and the set of reference points $\mathbf{R}$. Given a pairwise Gaussian kernel function $k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-||\mathbf{x}-\mathbf{y}||^2}{2h^2}\right)$, the relative error level $\epsilon > 0$, and the desired kernel sum $\Phi(\mathbf{q}; \mathbf{R}) = \sum_{\mathbf{r_j} \in \mathbf{R}} w_j k(\mathbf{q}, \mathbf{r_j})$ for each $\mathbf{q} \in \mathbf{Q}$,

Task: Compute an approximation $\widetilde{\Phi}(\mathbf{q}; \mathbf{R})$ for each $\mathbf{q} \in \mathbf{Q}$ such that $\left|\widetilde{\Phi}(\mathbf{q}; \mathbf{R}) - \Phi(\mathbf{q}; \mathbf{R})\right| \leq \epsilon \Phi(\mathbf{q}; \mathbf{R})$ as fast as possible.

Expansions in [84] and Chapter 3 require the computation of $\mathcal{O}(p^D)$ sub-terms. While effective in the context of computational physics problems, this is problematic in statistical/data mining applications, in which D may be larger than 2 or 3. Chapter 3 developed the translation operators and error bounds necessary to perform the original FGT-style $\mathcal{O}(p^D)$ approximation within the context of the dual-tree framework, demonstrating the first hierarchical fast Gauss transform. However, the new algorithm showed efficiency over any of the aforementioned methods over the entire range of bandwidths necessary in cross-validation, only in very low dimensions (3 or less). The Improved Fast Gauss Transform (IFGT) [201] introduced a rearranged series approximation requiring $\mathcal{O}(D^p)$ sub-terms, which seemed promising for higher

[1] This is a slightly generalized setting than the one in Chapter 3.

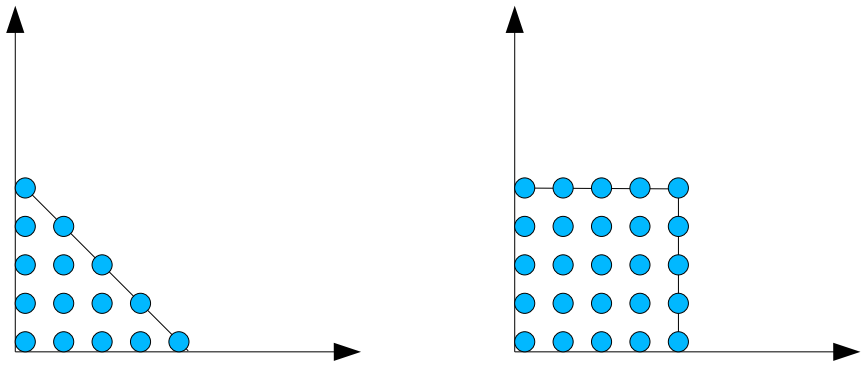


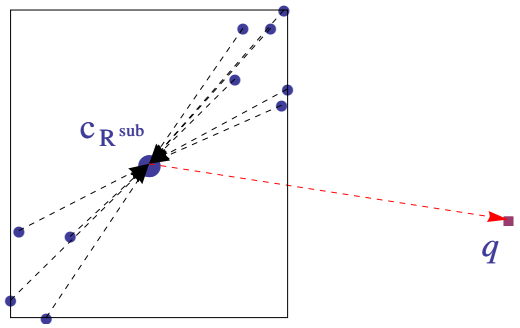
Figure 30: Take $D = 2$ and $p = 6$ for example. Left: $\mathcal{O}(D^p)$ (15 terms); Right: $\mathcal{O}(p^D)$ (25 terms) Think of "sampling" the Gaussian kernel at fixed basis functions. From bottom to top, left to right, we have multi-indices: $\mathcal{O}(D^p)$: (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (0, 1), (1, 1), (2, 1), (3, 1), (0, 2), (1, 2), (2, 2), (0, 3), (1, 3), (0, 4); $\mathcal{O}(p^D)$: (0, 0), (1, 0), (2, 0), (3, 0), (4, 0), (0, 1), (1, 1), (2, 1), (3, 1), (4, 1), (0, 2), (1, 2), (2, 2), (3, 2), (4, 2), (0, 3), (1, 3), (2, 3), (3, 3), (4, 3)

dimensions with an associated error bound, which was unfortunately incorrect. The IFGT was based on a flat set of clusters and did not provide any translation operators.

In this chapter, we demonstrate for the first time the $\mathcal{O}(D^p)$ (rather than $\mathcal{O}(p^D)$) expansion of the Gaussian kernel (different from that of the IFGT) within a *hierarchical* (dual-tree) algorithm. We also introduce a more efficient mechanism for automatically achieving the user's error tolerance which works with both discrete and continuous approximation schemes. We evaluate these new techniques empirically on real datasets, **revealing the strengths and weaknesses of the main current approaches for the first time**.

4.1 $\mathcal{O}(D^p)$ and $\mathcal{O}(p^D)$ Expansions

For concreteness, we first discuss the difference between $\mathcal{O}(p^D)$ and $\mathcal{O}(D^p)$ expansion. The $\mathcal{O}(p^D)$ expansion [84] utilizes the multiplicative nature of the Gaussian kernel (i.e. the multivariate isotropic Gaussian kernel is the product of univariate Gaussian kernels) and the univariate Taylor's theorem (Theorem 3.1.1). On the other hand, the $\mathcal{O}(D^p)$ expansion uses the multivariate Taylor's theorem.



(a)

{0, 0}	10.0
{1, 0}	3.59
{0, 1}	2.83
{2, 0}	13.7
{1, 1}	23.1
{0, 2}	11.3
{3, 0}	1.95
{2, 1}	4.23
{1, 2}	3.67
{0, 3}	1.16
{4, 0}	3.50
{3, 1}	11.2
{2, 2}	15.5
{1, 3}	10.6
{0, 4}	2.92

{0, 0}	10.0
{0, 1}	2.83
{0, 2}	11.3
{0, 3}	1.16
{0, 4}	2.92
{1, 0}	3.59
{1, 1}	23.1
{1, 2}	3.67
{1, 3}	10.6
{1, 4}	0.867
{2, 0}	13.7
{2, 1}	4.23
{2, 2}	15.5
{2, 3}	1.76
{2, 4}	4.08
{3, 0}	1.95
{3, 1}	11.2
{3, 2}	1.85
{3, 3}	5.16
{3, 4}	0.479
{4, 0}	3.50
{4, 1}	1.10
{4, 2}	3.93
{4, 3}	0.465
{4, 4}	1.03

(b)

Figure 31: In (b), the far-field moments using the reference points shown in (a) are computed in the $\mathcal{O}(D^p)$ expansion (left) and in the $\mathcal{O}(p^D)$ expansion (right) up to the same order $p = 5$; note that both representations store moments in a linear array representation, and the moments in the $\mathcal{O}(D^p)$ are a subset of those in the $\mathcal{O}(p^D)$ expansion of the same order.

Theorem 4.1.1. Multidimensional Taylor's Theorem: *Let* $\mathbf{O} \subset \mathbb{R}^D$ *be an open set. Let* $\mathbf{x}_* \in \mathbf{O}$ *and* f *be a function which is* n *times differentiable in* $\mathbf{O}$*. For any* $\mathbf{x} \in \mathbf{O}$*, there exists* $\theta \in \mathbb{R}$ *with* $0 < \theta < 1$ *such that* $f(\mathbf{x}) = \sum_{|\boldsymbol{\alpha}|<p} \frac{1}{\boldsymbol{\alpha}!} D^{\boldsymbol{\alpha}} f(\mathbf{x}_*)(\mathbf{x}-\mathbf{x}_*)^{\boldsymbol{\alpha}} + \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\boldsymbol{\alpha}!} D^{\boldsymbol{\alpha}} f(\mathbf{x}_* + \theta(\mathbf{x}-\mathbf{x}_*))(\mathbf{x}-\mathbf{x}_*)^{\boldsymbol{\alpha}}$*. The last term* $R_n = \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\boldsymbol{\alpha}!} D^{\boldsymbol{\alpha}} f(\mathbf{x}_* + \theta(\mathbf{x}-\mathbf{x}_*))(\mathbf{x}-\mathbf{x}_*)^{\boldsymbol{\alpha}}$ *is called the Lagrange remainder and* $|R_n| \leq \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\boldsymbol{\alpha}!} \sup_{0<\theta<1} \left| D^{\boldsymbol{\alpha}} f(\mathbf{x}_* + \theta(\mathbf{x}-\mathbf{x}_*)) \right| \prod_{d=1}^{D} \left| \mathbf{x}[d] - \mathbf{x}_*[d] \right|^{\boldsymbol{\alpha}[d]}$

Let us compare the far-field expansion and the local expansion under both the $\mathcal{O}(p^D)$ and $\mathcal{O}(D^p)$ expansion schemes.

Far-field Expansion: Let $\mathbf{R}^{sub}$ be a reference node. A *far-field expansion* for $\mathcal{O}(p^D)$ is given by:

$$\Phi(\mathbf{q_i}; \mathbf{R}^{sub}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \exp\left(\frac{-||\mathbf{q_i} - \mathbf{r_{j_n}}||^2}{2h^2}\right)$$

$$= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \prod_{d=1}^{D} \left(\sum_{\boldsymbol{\alpha}[d]=0}^{\infty} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \right)$$

$$= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \prod_{d=1}^{D} \left(\sum_{\boldsymbol{\alpha}[d]<p} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d)]}{\sqrt{2h^2}} \right) + \right.$$
$$\left. \sum_{\boldsymbol{\alpha}[d]\geq p} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \right)$$

Truncating after p terms along each dimension yields:

$$\Phi(\mathbf{q_i}; \mathbf{R}^{sub}) \approx \widetilde{\Phi}\left(\mathbf{q_i}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, \{\boldsymbol{\alpha} \in \left(\mathbb{Z}^+ \cup \{0\}\right)^D : \boldsymbol{\alpha} < p\}))\}\right)$$

$$= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \prod_{d=1}^{D} \left(\sum_{\boldsymbol{\alpha}[d]<p} \frac{1}{\boldsymbol{\alpha}[d]!} \left(\frac{\mathbf{r_{j_n}}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]} \left(\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}} \right) \right)$$

$$= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \sum_{\boldsymbol{\alpha}<p} \frac{1}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}} h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$$

$$= \sum_{\boldsymbol{\alpha}<p} \left[\sum_{\mathbf{r_{j_n}} \in R} \frac{w_{j_n}}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}} \right] h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$$

$$= \sum_{\boldsymbol{\alpha}<p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q_i} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$$

If the truncated part of the series per each dimension,

$$\sum_{\boldsymbol{\alpha}[d]\geq p} \frac{1}{\boldsymbol{\alpha}[d]!}\left(\frac{\mathbf{r_{j_n}}[d]-\mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}[d]} h_{\boldsymbol{\alpha}[d]}\left(\frac{\mathbf{q_i}[d]-\mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right)$$

is small, the truncation incurs small error. Ideally, we would like to choose the smallest p such that the truncation after the chosen order p incurs tolerable error; this will be discussed in Section 4.3. The transition from the fourth line to the fifth line follows from the definition of a multi-index $\boldsymbol{\alpha}$ to be less than a scalar p.

The $\mathcal{O}(D^p)$ expansion truncates the terms in a different way:

$$\begin{aligned}\Phi(\mathbf{q_i};\mathbf{R}^{sub}) \approx & \widetilde{\Phi}(\mathbf{q_i};\{(\mathbf{R}^{sub},\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}},\{\boldsymbol{\alpha}\in\left(\mathbb{Z}^+\cup\{0\}\right)^D:|\boldsymbol{\alpha}|<p\}))\}))\}) \\ &= \sum_{|\boldsymbol{\alpha}|<p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}})h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{q_i}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)\end{aligned}$$

Local Expansion: A *local expansion* is a Taylor expansion of the kernel sums about a representative point $\mathbf{c}_{\mathbf{Q}^{sub}}$ in a *query region* $\mathbf{Q}^{sub}$:

$$\begin{aligned}\Phi(\mathbf{q_i};\mathbf{R}^{sub}) &= \sum_{\mathbf{r_{j_n}}\in\mathbf{R}^{sub}} w_{j_n}\exp\left(\frac{-||\mathbf{q_i}-\mathbf{r_{j_n}}||^2}{2h^2}\right) \\ &= \sum_{\mathbf{r_{j_n}}\in\mathbf{R}^{sub}} w_{j_n}\prod_{d=1}^{D}\left(\sum_{n_d=0}^{\infty}\frac{(-1)^{n_d}}{n_d!}h_{n_d}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d]-\mathbf{r_{j_n}}[d]}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q_i}[d]-\mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}\right) \\ &= \sum_{\mathbf{r_{j_n}}\in\mathbf{R}^{sub}} w_{j_n}\prod_{d=1}^{D}\Bigg(\sum_{n_d<p}\frac{(-1)^{n_d}}{n_d!}h_{n_d}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d]-\mathbf{r_{j_n}}[d]}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q_i}[d]-\mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}+ \\ &\qquad \sum_{n_d\geq p}\frac{(-1)^{n_d}}{n_d!}h_{n_d}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}[d]-\mathbf{r_{j_n}}[d]}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q_i}[d]-\mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}\Bigg)\end{aligned}$$

Again, truncating after p terms along each dimension yields:

$$\begin{aligned}
&\widetilde{\Phi}(\mathbf{q_i}; \{(\mathbf{R}^{sub}, \mathsf{N}(\mathbf{c_{Q}}^{sub}, \{\boldsymbol{\alpha} \in (\mathbb{Z}^+ \cup \{0\})^D : \boldsymbol{\alpha} < p\}))\}) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \prod_{d=1}^{D} \left(\sum_{n_d < p} \frac{(-1)^{n_d}}{n_d!} h_{n_d} \left(\frac{\mathbf{c_{Q}}^{sub}[d] - \mathbf{r_{j_n}}[d]}{\sqrt{2h^2}} \right) \left(\frac{\mathbf{q_i}[d] - \mathbf{c_{Q}}^{sub}[d]}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}} \right) \\
&= \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} w_{j_n} \sum_{\boldsymbol{\beta} < p} \frac{(-1)^{\boldsymbol{\beta}}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}} \left(\frac{\mathbf{c_{Q}}^{sub} - \mathbf{r_{j_n}}}{\sqrt{2h^2}} \right) \left(\frac{\mathbf{q_i} - \mathbf{c_{Q}}^{sub}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}} \\
&= \sum_{\boldsymbol{\beta} < p} \left[\sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{(-1)^{\boldsymbol{\beta}} w_{j_n}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}} \left(\frac{\mathbf{c_{Q}}^{sub} - \mathbf{r_{j_n}}}{\sqrt{2h^2}} \right) \right] \left(\frac{\mathbf{q_i} - \mathbf{c_{Q}}^{sub}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}} \\
&= \sum_{\boldsymbol{\beta} < p} N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c_{Q}}^{sub}, \{\boldsymbol{\beta} \in (\mathbb{Z}^+ \cup \{0\})^D : \boldsymbol{\beta} < p\}))\}) \left(\frac{\mathbf{q_i} - \mathbf{c_{Q}}^{sub}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}
\end{aligned}$$

The truncation incurs small error, given that the absolute value of the truncated part:

$$\sum_{n_d \geq p} \frac{(-1)^{n_d}}{n_d!} h_{n_d} \left(\frac{\mathbf{c_{Q}}^{sub}[d] - \mathbf{r_{j_n}}[d]}{\sqrt{2h^2}} \right) \left(\frac{\mathbf{q_i}[d] - \mathbf{c_{Q}}^{sub}[d]}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}$$

is bounded by a small quantity. The error bound criterion will again be discussed in Section 4.3. The $\mathcal{O}(D^p)$ expansion of the local expansion is given by:

$$\begin{aligned}
\Phi(\mathbf{q_i}; \mathbf{R}^{sub}) \approx &\widetilde{\Phi}(\mathbf{q_i}; \{(\mathbf{R}^{sub}, \mathsf{N}(\mathbf{c_{Q}}^{sub}, \{\boldsymbol{\beta} \in \left(\mathbb{Z}^+ \cup \{0\}\right)^D : |\boldsymbol{\beta}| < p\}))\})\}) \\
&= \sum_{|\boldsymbol{\beta}| < p} N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c_{Q}}^{sub}, \{\boldsymbol{\beta} \in \left(\mathbb{Z}^+ \cup \{0\}\right)^D : |\boldsymbol{\beta}| < p\}))\}) \left(\frac{\mathbf{q_i} - \mathbf{c_{Q}}^{sub}}{\sqrt{2h^2}} \right)^{\boldsymbol{\beta}}
\end{aligned}$$

4.2 Translation Operators

Since the properties of the Gaussian kernel do not require that approximation be made in the local fashion, the original FGT used a flat grid with only **far-to-local** operator whose associated incorrect error was corrected by [11]. [114] derived two additional translation operators necessary for a *hierarchical* FGT and the associated error bounds for $\mathcal{O}(p^D)$ expansion of Hermite/Taylor coefficients. For a review of all three translation operators, see Chapter 3.

4.3 Error Bounds for $\mathcal{O}(D^p)$ Expansions

Because Hermite/Taylor expansions are truncated after a finite number of terms, we incur an error in approximation. In order to bound the total approximation error, we need one error bound for each translation operator. In [114], the Hermite and the Taylor expansion were treated as products of D univariate Hermite/Taylor expansions. The trailing sum in each univariate expansion was bounded using the property of infinite geometric series, which in turn limited the size of the query/reference node for pruning to be valid. Here, we use the same translation operators, but instead view each expansion as a *vector* function and use the $\mathcal{O}(D^p)$ expansion advocated in [201]. The new error bounds based on this new expansion scheme depend on the multidimensional Taylor's Theorem, and effectively eliminate the node size restriction imposed by the $\mathcal{O}(p^D)$ expansion [84, 114].

The first lemma gives an upper bound on the absolute error on estimating a reference node contribution by evaluating a truncated Hermite expansion. The second lemma gives an upper bound on the absolute error incurred from approximating the contribution of a reference node by evaluating the Taylor series formed via direct local accumulation of each reference point.

Lemma 4.3.1. *Suppose we are given a far-field expansion of* $\mathbf{R}^{sub}$ *about* $\mathbf{c}_{\mathbf{R}^{sub}}$*:*
$\widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\}) = \sum_{|\boldsymbol{\alpha}| \leq p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}} \left(\frac{\mathbf{q} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)$ *where*
$M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) = \sum_{\mathbf{r_{j_n}} \in \mathbf{R}^{sub}} \frac{w_{j_n}}{\boldsymbol{\alpha}!} \left(\frac{\mathbf{r_{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}}$. *Then:* $|\widetilde{\Phi}(\mathbf{q}; \{(\mathbf{R}^{sub}, \mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}}, p))\})) -$
$\Phi(\mathbf{q}; \mathbf{R}^{sub})| \leq E_{\mathsf{F}}(p) = W(\mathbf{R}^{sub}) \frac{\exp\left(\frac{-d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})^2}{4h^2} \right) \binom{D+p-1}{D-1} r_{\mathbf{R}^{sub}}^p}{\sqrt{\left(\lfloor \frac{p}{D} \rfloor ! \right)^{D-p'} \left(\lceil \frac{p}{D} \rceil ! \right)^{p'}}}$ *where*
$r_{\mathbf{R}^{sub}} = \max_{\mathbf{r_j} \in \mathbf{R}^{sub}} \frac{||\mathbf{r_j} - \mathbf{c}_{\mathbf{R}^{sub}}||_\infty}{h}$ *and* $p' = p \mod D$ *and* $W(\mathbf{R}^{sub}) = \sum_{\mathbf{r_j} \in \mathbf{R}^{sub}} w_j$.

Proof. By Theorem 4.1.1 and the triangle inequality,

$$\left|\Phi(\mathbf{q};\mathbf{R}^{sub}) - \sum_{|\boldsymbol{\alpha}|<p} M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{q}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)\right|$$

$$\leq \sum_{\mathbf{r_j}\in\mathbf{R}^{sub}} w_r \left| k(||\mathbf{q}-\mathbf{r_j}||) - \sum_{|\boldsymbol{\alpha}|<p} \frac{1}{\boldsymbol{\alpha}!} h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{q}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{r_j}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}}\right|$$

$$\leq W(\mathbf{R}^{sub}) \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\boldsymbol{\alpha}!} \max_{\mathbf{q}\in\mathbf{Q}^{sub},\mathbf{r_j}\in\mathbf{R}^{sub}} \left| h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{q}-\mathbf{r_j}}{\sqrt{2h^2}}\right)\right| \prod_{d=1}^{D} \left|\frac{\mathbf{r_j}[d]-\mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\alpha}[d]}$$

$$\leq W(\mathbf{R}^{sub}) \exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right) \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\boldsymbol{\alpha}!}\left(\sqrt{2}\right)^{\boldsymbol{\alpha}} \sqrt{\boldsymbol{\alpha}!} \prod_{d=1}^{D} \left|\frac{\mathbf{r_j}[d]-\mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\alpha}[d]}$$

$$\leq W(\mathbf{R}^{sub}) \exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right) \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\sqrt{\boldsymbol{\alpha}!}} \prod_{d=1}^{D} \left|\frac{\mathbf{r_j}[d]-\mathbf{c}_{\mathbf{R}^{sub}}[d]}{h}\right|^{\boldsymbol{\alpha}[d]}$$

$$\leq W(\mathbf{R}^{sub}) \exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right) \sum_{|\boldsymbol{\alpha}|=p} \frac{r^{\boldsymbol{\alpha}}_{\mathbf{R}^{sub}}}{\sqrt{\boldsymbol{\alpha}!}} \leq W(\mathbf{R}^{sub}) \frac{\exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right)\binom{D+p-1}{D-1} r^p_{\mathbf{R}^{sub}}}{\sqrt{\left(\lfloor\frac{p}{D}\rfloor!\right)^{D-p'}\left(\lceil\frac{p}{D}\rceil!\right)^{p'}}}$$

□

Lemma 4.3.2. *Suppose we are given the local expansion about* $\mathbf{c}_{\mathbf{Q}^{sub}}$ *of the given query node* $\mathbf{Q}^{sub}$ *accounting for the kernel sum contribution of* $\mathbf{R}^{sub}$*:* $\widetilde{\Phi}(q_{i_m};\{(\mathbf{R}^{sub},\mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}},p))\}) = \sum_{|\boldsymbol{\beta}|\leq p} N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub},(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})\left(\frac{\mathbf{q_{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$ *where* $\mathbf{q_{i_m}}\in\mathbf{Q}^{sub}$ *and* $N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub},(\mathbf{c}_{\mathbf{Q}^{sub}},p))\}) = \sum_{\mathbf{r_{j_n}}\in\mathbf{R}^{sub}} \frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!} h_{\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}-r_{j_n}}{\sqrt{2h^2}}\right)$ *Then:* $\left|\widetilde{\Phi}(\mathbf{q_{i_m}};\{(\mathbf{R}^{sub},\mathsf{N}(\mathbf{c}_{\mathbf{Q}^{sub}},p))\}) - \Phi(\mathbf{q_{i_m}};\mathbf{R}^{sub})\right| \leq E_{\mathsf{N}}(p) = W(\mathbf{R}^{sub}) \frac{\exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right)\binom{D+p-1}{D-1} r^p_{\mathbf{Q}^{sub}}}{\sqrt{\left(\lfloor\frac{p}{D}\rfloor!\right)^{D-p'}\left(\lceil\frac{p}{D}\rceil!\right)^{p'}}}$ *where* $r_{\mathbf{Q}^{sub}} = \max_{\mathbf{q_i}\in\mathbf{Q}^{sub}} \frac{||\mathbf{q_i}-\mathbf{c}_{\mathbf{Q}^{sub}}||_\infty}{h}$ *and* $p' = p \mod D$.

Proof. The derivation is similar to one in Lemma 4. □

The final lemma gives an upper bound on the absolute error incurred by approximating the reference node contribution by the Taylor expansion converted from the truncated Hermite expansion.

Lemma 4.3.3. *A truncated far-field expansion centered about* $\mathbf{c}_{\mathbf{R}^{sub}}$ *of* $\mathbf{R}^{sub}$,

$$\widetilde{\Phi}(\mathbf{q};\{(\mathbf{R}^{sub},\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}},p))\})=\sum_{\boldsymbol{\alpha}\leq p}M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}})h_{\boldsymbol{\alpha}}\left(\frac{\mathbf{q}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)$$

has the following local expansion about $\mathbf{c}_{\mathbf{Q}^{sub}}$ *of* $\mathbf{Q}^{sub}$ *for* $\mathbf{q}_{\mathbf{i_m}}\in\mathbf{Q}^{sub}$*:*
$\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(\mathbf{R}^{sub},\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}},p))\})=\sum_{\boldsymbol{\beta}\geq 0}N_{\boldsymbol{\beta}}(\{(M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$ *where:*
$N_{\boldsymbol{\beta}}(\{(M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})=\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}\sum_{|\boldsymbol{\alpha}|\leq p}M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}})h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)$.
Let $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})=\sum_{|\boldsymbol{\beta}|\leq p}\widetilde{N}_{\boldsymbol{\beta}}(\{(M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})$
$\left(\frac{\mathbf{q}_{\mathbf{i_m}}-\mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$*, a truncation of the local expansion of* $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(\mathbf{R}^{sub},\mathsf{F}(\mathbf{c}_{\mathbf{R}^{sub}},p))\})$ *after* $\mathcal{O}(D^p)$ *terms. Then:*

$$\left|\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}};\{(M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}}),\widetilde{\mathsf{N}}(\mathbf{c}_{\mathbf{Q}^{sub}},p))\})-\Phi(\mathbf{q};\mathbf{R}^{sub})\right|\leq E_{\widetilde{\mathsf{N}}}(p)$$

$$=W(\mathbf{R}^{sub})\frac{\exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right)\binom{D+p-1}{D-1}}{\sqrt{\left(\lfloor\frac{p}{D}\rfloor!\right)^{D-p'}\left(\lceil\frac{p}{D}\rceil!\right)^{p'}}}\left(r_{\mathbf{Q}^{sub}}^p+\left(\sqrt{2}r_{\mathbf{R}^{sub}}\right)^p\binom{D+p-1}{D}\left(\sqrt{2}r_{\mathbf{Q}^{sub}}\right)^{I(\sqrt{2}r_{\mathbf{Q}^{sub}})}\right)$$

where $r_{\mathbf{Q}^{sub}}=\max_{\mathbf{q_i}\in\mathbf{Q}^{sub}}\frac{||\mathbf{q_i}-\mathbf{q_i}||_\infty}{h}$,

$r_{\mathbf{R}^{sub}}=\max_{\mathbf{r_j}\in\mathbf{R}^{sub}}\frac{||\mathbf{r_j}-\mathbf{c}_{\mathbf{R}^{sub}}||_\infty}{h}$, $p'=p \mod D$ *and* $I(x)=\begin{cases}0, & 0\leq x\leq 1\\ p-1, & otherwise\end{cases}$.

Proof. Let $E_1=\sum_{|\boldsymbol{\beta}|<p}\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}\sum_{|\boldsymbol{\alpha}|\geq p}\frac{1}{\boldsymbol{\alpha}!}\left(\frac{\mathbf{r_j}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}}h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{q_i}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q_i}-\mathbf{q_i}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$ and
$E_2=\sum_{|\boldsymbol{\beta}|\geq p}\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}h_{\boldsymbol{\beta}}\left(\frac{\mathbf{q_i}-\mathbf{r_j}}{\sqrt{2h^2}}\right)\left(\frac{\mathbf{q_i}-\mathbf{q_i}}{\sqrt{2h^2}}\right)^{\boldsymbol{\beta}}$ Then,

$$\left|\Phi(\mathbf{q_i})-\sum_{|\boldsymbol{\beta}|<p}\frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}\sum_{|\boldsymbol{\alpha}|<p}M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub},\mathbf{c}_{\mathbf{R}^{sub}})h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{q_i}-\mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}\right)\right|\leq W(\mathbf{R}^{sub})(|E_1|+|E_2|)$$

Clearly, $|E_2| \leq \frac{\exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right)\binom{D+p-1}{D-1} r_{\mathbf{Q}^{sub}}^p}{\sqrt{\left(\lfloor \frac{p}{D} \rfloor !\right)^{D-p'}\left(\lceil \frac{p}{D} \rceil !\right)^{p'}}}$. In addition,

$$|E_1| \leq \sum_{|\boldsymbol{\beta}|<p} \frac{1}{\boldsymbol{\beta}!} \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\boldsymbol{\alpha}!} \prod_{d=1}^{D} \left|\frac{\mathbf{r_j}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\alpha}[d]} \left|\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\beta}[d]} \max_{\substack{\mathbf{q_i}\in\mathbf{Q}^{sub} \\ \mathbf{r_j}\in\mathbf{R}^{sub}}} \left| h_{\boldsymbol{\alpha}+\boldsymbol{\beta}}\left(\frac{\mathbf{q_i}-\mathbf{r_j}}{\sqrt{2h^2}}\right)\right|$$

$$\leq \exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right) \sum_{|\boldsymbol{\beta}|<p} \frac{1}{\sqrt{\boldsymbol{\beta}!}} \sum_{|\boldsymbol{\alpha}|=p} \sqrt{\frac{(\boldsymbol{\alpha}+\boldsymbol{\beta})!}{\boldsymbol{\alpha}!\boldsymbol{\beta}!}} \frac{\sqrt{2}^{\boldsymbol{\alpha}+\boldsymbol{\beta}}}{\sqrt{\boldsymbol{\alpha}!}}$$
$$\prod_{d=1}^{D} \left|\frac{\mathbf{r_j}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\alpha}[d]} \left|\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\beta}[d]}$$

$$\leq \exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right) \sum_{|\boldsymbol{\beta}|<p} \frac{1}{\sqrt{\boldsymbol{\beta}!}} \sum_{|\boldsymbol{\alpha}|=p} \sqrt{2}^{\boldsymbol{\alpha}+\boldsymbol{\beta}} \frac{\sqrt{2}^{\boldsymbol{\alpha}+\boldsymbol{\beta}}}{\sqrt{\boldsymbol{\alpha}!}}$$
$$\prod_{d=1}^{D} \left|\frac{\mathbf{r_j}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\alpha}[d]} \left|\frac{\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d]}{\sqrt{2h^2}}\right|^{\boldsymbol{\beta}[d]}$$

$$\leq \exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right) \sum_{|\boldsymbol{\beta}|<p} \frac{1}{\sqrt{\boldsymbol{\beta}!}} \sum_{|\boldsymbol{\alpha}|=p} \frac{1}{\sqrt{\boldsymbol{\alpha}!}}$$
$$\prod_{d=1}^{D} \left|\frac{\sqrt{2}(\mathbf{r_j}[d] - \mathbf{c}_{\mathbf{R}^{sub}}[d])}{h}\right|^{\boldsymbol{\alpha}[d]} \left|\frac{\sqrt{(}2)(\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d])}{h}\right|^{\boldsymbol{\beta}[d]}$$

$$\leq \frac{\exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right)\binom{D+p-1}{D-1}\left(\sqrt{2}r_{\mathbf{R}^{sub}}\right)^p}{\sqrt{\left(\lfloor \frac{p}{D} \rfloor !\right)^{D-p'}\left(\lceil \frac{p}{D} \rceil !\right)^{p'}}} \sum_{|\boldsymbol{\beta}|<p} \frac{1}{\sqrt{\boldsymbol{\beta}!}} \prod_{d=1}^{D} \left|\frac{\sqrt{(}2)(\mathbf{q_i}[d] - \mathbf{c}_{\mathbf{Q}^{sub}}[d])}{h}\right|^{\boldsymbol{\beta}[d]}$$

$$\leq \frac{\exp\left(\frac{-d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})^2}{4h^2}\right)\binom{D+p-1}{D-1}}{\sqrt{\left(\lfloor \frac{p}{D} \rfloor !\right)^{D-p'}\left(\lceil \frac{p}{D} \rceil !\right)^{p'}}} \left(\sqrt{2}r_{\mathbf{R}^{sub}}\right)^p \binom{D+p-1}{D} \left(\sqrt{2}r_{\mathbf{Q}^{sub}}\right)^{I(\sqrt{2}r_{\mathbf{Q}^{sub}})}$$

□

4.4 New Error Guarantee Rule

We now specify the function CanSummarize($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$), which has only local information (contained in the query node $\mathbf{Q}^{sub}$ and the reference node $\mathbf{R}^{sub}$) available to it. In the dual-tree finite-difference algorithm (DFD) [77], the function Summarize($\mathbf{Q}^{sub}$,$\mathbf{R}^{sub}$) approximates the contribution of $\mathbf{R}^{sub}$ to each query point $\mathbf{q_i}$ in $\mathbf{Q}^{sub}$, $\Phi(\mathbf{q_i}; \mathbf{R}^{sub})$, by $\widetilde{\Phi}(\mathbf{q_i}; \mathbf{R}^{sub}) = W(\mathbf{R}^{sub})\bar{K} = W(\mathbf{R}^{sub})\frac{k(d^u(\mathbf{Q}^{sub},\mathbf{R}^{sub}))+k(d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub}))}{2}$

where $W(\mathbf{R}^{sub}) = \sum_{\mathbf{r_j} \in \mathbf{R}^{sub}} w_j$ and $d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$ and $d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$ are lower and upper bounds on the distance between $\mathbf{q_i} \in \mathbf{Q}^{sub}$ and $\mathbf{r_j} \in \mathbf{R}^{sub}$, respectively. These distances are easily obtained using the bounding boxes of the nodes. By using these bounds DFD algorithm maintains a running lower bound $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})$ on $\Phi(\mathbf{q_i}; \mathbf{R}^{sub})$ which holds for all $\mathbf{q_i} \in \mathbf{Q}^{sub}$. In Section 4.3, we laid out more approximation methods in addition to finite-difference approximation: evaluating a truncated Hermite expansion centered at $\mathbf{c}_{\mathbf{R}^{sub}}$, forming a truncated Taylor expansion centered at $\mathbf{q_i}$ using each reference point, and forming an approximated truncated Taylor expansion centered at $\mathbf{q_i}$ by converting the truncated Hermite expansion centered at $\mathbf{c}_{\mathbf{R}^{sub}}$. This change was described in Section 3.1.8.

Our series-expansion based algorithm uses four different approximation methods, i.e. $\mathsf{A} \in \{\mathsf{D}, \widetilde{\mathsf{N}}(\mathbf{c}, p), \mathsf{F}(\mathbf{c}, p), \mathsf{N}(\mathbf{c}, p)\}$. For each $\mathbf{R}^{sub}$, an approximation method is chosen. D denotes the exhaustive computation of $\sum_{\mathbf{r_{j_n}} \in \mathbf{R}} w_{j_n} k(||\mathbf{q_i} - \mathbf{r_{j_n}}||)$. $\widetilde{\mathsf{N}}(\mathbf{c}, p)$ denotes the translation of the order p far-field moments of $\mathbf{R}^{sub}$ to the local moments in the query node $\mathbf{Q}^{sub}$ that owns $\mathbf{q_i}$ about a representative centroid $\mathbf{c}$ inside $\mathbf{Q}^{sub}$. $\mathsf{F}(\mathbf{c}, p)$ denotes the evaluation up to the p-th order far-field expansion formed by the moments of $\mathbf{R}^{sub}$ expanded about a representative point $\mathbf{c}$ inside $\mathbf{R}^{sub}$. $\mathsf{N}(\mathbf{c}, p)$ denotes the p-th order direct accumulation of the local moments due to $\mathbf{R}^{sub}$ about a representative centroid $\mathbf{c}$ inside $\mathbf{Q}^{sub}$ that owns $\mathbf{q_i}$.

Now note that: $E_{\mathsf{D}} = 0$, and E_{F}, E_{N}, and $E_{\widetilde{\mathsf{N}}}$ are given as Lemma 4.3.1, 4.3.2, 4.3.3 respectively. The approximation rule above essentially gives each reference node $\mathbf{R}^{sub}$ a maximum relative error proportional to the sum of the weights of reference points it contains. In considering the i-th reference node contribution, when $\mathsf{A} = \mathsf{D}$ (exhaustive direct method), the maximum allowable relative error of $\frac{W(\mathbf{R_i})\epsilon}{W(\mathbf{R})}$ is not used up; Otherwise, if $\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R}) > 0$, pruning requires only a relative error of $\frac{W'(\mathbf{R_i})\epsilon}{W(\mathbf{R})}$ where $W'(\mathbf{R_i}) = \frac{W(\mathbf{R})E_{\mathsf{A_i}}}{\epsilon\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})}$. Our new approximation rule notes that the portion of the weights not used to cover the incurred pruning error can be stored into a field

variable of $\mathbf{Q}^{sub}$ (initialized to zero before the computation and denoted $\Delta^T(\mathbf{Q}^{sub})$ hereon) in each query node $\mathbf{Q}^{sub}$ to use them in future pruning opportunities. The first case yields $W(\mathbf{R_i})$ as the leftover, while the second case (pruned case) yields $W(\mathbf{R_i}) - W'(\mathbf{R_i})$.

Given $\mathsf{A_i} \in \mathsf{A}$ with the maximum absolute error of $E_{\mathsf{A_i}}$, we now modify the approximation condition to: $\frac{E_{\mathsf{A_i}}}{\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})} \leq \frac{\epsilon(W(\mathbf{R}^{sub}) + W_T)}{W}$. Solving for W_T yields: $W_T \geq W(\mathbf{R}^{sub})\big(\frac{WE_{\mathsf{A_i}}}{\epsilon\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})} - 1\big)$. Whenever a pruning is attempted, the modified algorithm will evaluate the right handside of the inequality. If the evaluated value is negative, it represents the leftover "token" after pruning is performed and $\Delta^T(\mathbf{Q}^{sub})$ of the current query node will be incremented by $W(\mathbf{R}^{sub})\big(1 - \frac{W(\mathbf{R})E_{\mathsf{A_i}}}{\epsilon\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})}\big)$. If positive, it represents the required extra "token" from the $\Delta^T(\mathbf{Q}^{sub})$ slot of the current query node, in order to prune the given query and reference node pair. If $\Delta^T(\mathbf{Q}^{sub}) \geq W_T$, pruning succeeds and $\Delta^T(\mathbf{Q}^{sub})$ is decremented by $W(\mathbf{R}^{sub})\big(\frac{W(\mathbf{R})E_{\mathsf{A_i}}}{\epsilon\Phi^l(\mathbf{Q}^{sub} \times \mathbf{R})} - 1\big)$.

4.5 New Dual-tree Algorithm

During the preprocessing phase, the Hermite moments of order *PLIMIT* is precomputed for the reference tree. For the experimental results, we have fixed $PLIMIT = 8$ for $D = 2$, $PLIMIT = 6$ for $D = 3$, $PLIMIT = 4$ for $D = 5$, $PLIMIT = 2$ for $D = 6$. We presume that $PLIMIT = 1$ for $D > 6$.

During the recursive function call, an optimized version of finite-difference pruning is first attempted. In case of failure, we attempt FMM-type pruning in which we choose the cheapest operation given a query node $\mathbf{Q}^{sub}$ and a reference node $\mathbf{R}^{sub}$ from the followings: direct Hermite evaluation, direct local accumulation, H2L translation, and exhaustive computations. Roughly, direct Hermite evaluations at each $\mathbf{q_i} \in \mathbf{Q}^{sub}$ is $\mathcal{O}(|\mathbf{Q}^{sub}|D^{p_{DH}+1})$, direct local accumulation $\mathcal{O}(|\mathbf{R}^{sub}|D^{p_{DL}+1})$, H2L translation $\mathcal{O}(D^{2p_{H2L}+1})$, an exhaustive method $\mathcal{O}(D|\mathbf{Q}^{sub}||\mathbf{R}^{sub}|)$. In our algorithm, if an exhaustive method is selected, we let the recursion continue, hoping pruning can

occur in the finer level of recursion. It is possible to hand-tune the exact cutoffs for determining the optimal choice, but these rough approximations seem to work well.

In the post-processing step, we perform a breadth-first traversal of the query tree. The algorithm is similar to the one described in Chapter 3.

4.6 Experiments and Conclusions

We empirically evaluated the runtime performance of six algorithms on six real-world datasets (astronomy (2-D), physical simulation (3-D), pharmaceutical (5-D), biology (7-D), forestry (10-D), image textures (16-D)) scaled to fit in $[0,1]^D$ hypercube, for kernel density estimation at every query point with a range of bandwidths, from 3 orders of magnitude smaller than optimal to three orders larger than optimal, according to the standard least-squares cross-validation scores [170]. In our case, the set of reference points is the same as the set of query points. All datasets have 50K points so that the exact exhaustive method can be tractably computed. We set the tolerance $\epsilon = 0.01$. We compare: **FGT** (Fast Gauss Transform [84]), **IFGT** (Improved Fast Gauss Transform [201]), **DFD** (dual-tree with finite-difference [77]), **DFDO** (dual-tree with finite-difference and improved error control (Section 3.2)), **DFTO** (dual-tree with $\mathcal{O}(p^D)$ expansion [114] and improved error control), and **DITO** (dual-tree with $\mathcal{O}(D^p)$ expansion and improved error control). All times (which include preprocessing but exclude parameter selection time) are in CPU seconds on a dual Intel Xeon 3 GHz with 2 Gb of main memory/1 Mb of CPU cache[2]. Codes are in C/C++, compiled under $-\mathbf{O6}\ -\mathbf{funroll-loops}$ flags on Linux kernel 2.6.9-11. The measurements in columns two to eight are obtained by running the algorithms at the bandwidth kh^* where $10^{-3} \leq k \leq 10^3$ is the constant in the corresponding column. The dual-tree algorithms all achieve the error tolerance automatically. We also note that the **FGT** uses a different error tolerance definition: $|\widetilde{\Phi}(\mathbf{q_i}) - \Phi(\mathbf{q_i})| \leq W\tau$. We first set $\tau = \epsilon$,

[2]Experimental setups are the same as [112].

Table 3: Speedup results on the 2-D, the 3-D, and the 5-D datasets.

$sj2-50000-2, D=2, N=50000, h^*=0.00139506$								
$Alg \backslash h^*$	10^{-3}	10^{-2}	10^{-1}	1	10^1	10^2	10^3	**Σ**
Naive	452	452	452	452	452	452	452	**3164**
FGT	X	X	X	4.36	1.66	0.26	0.13	**X**
IFGT	∞	∞	∞	∞	∞	∞	7.05	∞
DFD	1.98	3.12	2.2	8.12	85.6	230	1.99	**333**
DFDO	2.02	3.18	2.19	7.08	77.7	170	0.82	**263**
DFTO	2.05	3.22	2.27	7.44	5.37	2.49	0.72	**23.6**
DITO	2.61	3.88	3.00	9.21	7.64	1.51	0.84	**28.7**
$mockgalaxy-D-1M-rnd, D=3, N=50000, h^*=0.000768201$								
$Alg \backslash h^*$	10^{-3}	10^{-2}	10^{-1}	1	10^1	10^2	10^3	**Σ**
Naive	461	461	461	461	461	461	461	**3227**
FGT	X	X	X	X	∞	∞	∞	**X**
IFGT	∞	∞	∞	∞	∞	∞	∞	∞
DFD	1.37	1.40	1.32	0.96	1.29	57.6	552	**616**
DFDO	1.40	1.43	1.35	0.97	1.25	44.5	355	**406**
DFTO	1.45	1.48	1.41	1.03	1.37	20	28.3	**55**
DITO	2.29	2.32	2.28	1.92	2.28	40.6	8.65	**60.3**
$bio5-rnd, D=5, N=50000, h^*=0.000567161$								
$Alg \backslash h^*$	10^{-3}	10^{-2}	10^{-1}	1	10^1	10^2	10^3	**Σ**
Naive	491	491	491	491	491	491	491	**3437**
FGT	X	X	X	X	X	X	X	**X**
IFGT	∞	∞	∞	∞	∞	∞	∞	∞
DFD	5.59	6.49	13.5	17.1	128	577	169	**917**
DFDO	5.75	6.67	13.7	16.2	113	544	81.6	**781**
DFTO	5.80	6.70	13.8	16.5	123	422	282	**870**
DITO	6.92	7.86	15.6	19.3	133	365	6.10	**554**

halving it until the error tolerance ϵ was met. For the **IFGT**, we created an automatic scheme to tweak its multiple parameters based on recommendations given in the paper and software documentation: For $D = 2$, use $p = 8$; for $D = 3$, use $p = 6$; set $\rho_x = 2.5$; start with $K = \sqrt{N}$ and double K until the error tolerance is met. When this failed to meet the tolerance, we resorted to additional trial and error by hand. We are primarily concerned with the sum of the times over all the bandwidths, shown in the last column of the table. Entries in the tables of 'X' denote cases where the algorithm exhausted RAM and caused a segmentation fault. Entries of ∞ denote cases where no setting of the algorithm's parameters was able to satisfy the error tolerance.

Our results demonstrate that the $\mathcal{O}(D^p)$ expansion helps reduce the computational time on datasets of dimensionality up to 5. For example, on the 2-D dataset, the new algorithm **DITO** performed about 12 times as fast as the original **DFD** algorithm, which is in itself an improvement over the naive algorithm. The datasets above five dimensions, however, present difficulty for the series expansion idea to be effective, and the new algorithm is slower than **DFD** algorithm. Yet the algorithm with the optimized pruning rule (**DFDO**) consistenyl yields about 10 % to 15 % improvement over **DFD** algorithm in higher dimensions.

Table 4: Speedup results on the 7-D, the 10-D, and the 16-D datasets.

$pall7 - rnd, D = 7, N = 50000, h^* = 0.00131865$								
$Alg \backslash h^*$	10^{-3}	10^{-2}	10^{-1}	1	10^1	10^2	10^3	$\mathbf{\Sigma}$
Naive	511	511	511	511	511	511	511	**3577**
FGT	X	X	X	X	X	X	X	**X**
IFGT	∞	∞	∞	∞	∞	∞	∞	∞
DFD	14.9	15.1	16.6	37.7	50.8	372	625	**1132**
DFDO	15.5	15.6	17.3	38.2	49	321	587	**1044**
DFTO	15.6	15.6	17.4	38.4	50.2	337	621	**1095**
DITO	16.5	16.7	18.4	40.5	54.7	362	703	**1212**
$covtype - rnd, D = 10, N = 50000, h^* = 0.0154758$								
$Alg \backslash h^*$	10^{-3}	10^{-2}	10^{-1}	1	10^1	10^2	10^3	$\mathbf{\Sigma}$
Naive	515	515	515	515	515	515	515	**3605**
FGT	X	X	X	X	X	X	X	**X**
IFGT	∞	∞	∞	∞	∞	∞	∞	∞
DFD	26.5	29.7	88.2	104	557	659	11.4	**1476**
DFDO	27.2	30.5	90.2	98.2	515	623	5.73	**1390**
DFTO	27.4	30.7	90.6	101	477	660	6.10	**1393**
DITO	28.4	31.6	92.8	106	490	668	6.19	**1423**
$CoocTexture - rnd, D = 16, N = 50000, h^* = 0.0263958$								
$Alg \backslash h^*$	10^{-3}	10^{-2}	10^{-1}	1	10^1	10^2	10^3	$\mathbf{\Sigma}$
Naive	558	558	558	558	558	558	558	**3906**
FGT	X	X	X	X	X	X	X	**X**
IFGT	∞	∞	∞	∞	∞	∞	∞	∞
DFD	19.3	36.6	107	199	611	641	0.56	**1614**
DFDO	19.9	36.4	107	237	589	375	0.58	**1365**
DFTO	20.1	37.8	108	189	629	401	0.60	**1386**
DITO	26.2	38.9	112	196	655	437	0.62	**1466**

CHAPTER V

MONTE CARLO MULTIPOLE METHOD

In this chapter, we continue exploring the problem of accelerating the Gaussian kernel sums (Equation (3.0.2)). In Chapter 3 and Chapter 4, we advocated the usage of the most successful class of acceleration methods that employ "higher-order divide and conquer" or *generalized N-body algorithms (GNA)* [78]. This approach can use any spatial partioning tree such as *kd*-trees or ball-trees for both the query set $\mathbf{Q}$ and reference data $\mathbf{R}$ and performs a simulataneous recursive descent on both trees.

GNA with relative error bounds (Definition 2.4.2) utilized bounding boxes and additional *cached-sufficient statistics* such as higher-order moments needed for series-expansion. The original framework [78, 82, 77, 80] utilized bounding-box based error bounds which tend to be very loose, which resulted in slow empirical performance around suboptimally small and large bandwidths. The most recent extesnsions described in Chapter 3 and Chapter 4 extended GNA-based Gaussian summations with series-expansion which provided tighter bounds; it showed enormous performance improvements, but only up to low dimensional settings (up to $D = 5$) since the number of required terms in series expansion increases exponentially with respect to D.

[95] introduces an iterative sampling based GNA for accelerating the computation of nested sums (a related easier problem). Its speedup is achieved by replacing pessimistic error bounds provided by bounding boxes with normal-based confidence interval from Monte Carlo sampling. [95] demonstrates the speedup many orders of magnitude faster than the previous state of the art in the context of computing aggregates over the queries (such as the LSCV score for selecting the optimal bandwidth). However, the authors did not discuss the sampling-based approach for computations

that require *per-query* estimates, such as those required for kernel density estimation.

None of the previous approaches for kernel summations addresses the issue of reducing the computational cost of each distance computation which incurs $\mathcal{O}(D)$ cost. However, the *intrinsic dimensionality* d of most high-dimensional datasets is much smaller than the explicit dimension D (that is, $d \ll D$). [121] proposed tree structures using a global dimension reduction method, such as random projection, as a preprocessing step for efficient $(1+\epsilon)$ approximate nearest neighbor search. Similarly, we develop a new data structure for kernel summations; our new data structure is constructed in a top-down fashion to perform the initial spatial partitioning in the original input space $\mathbb{R}^D$ and performs a local dimension reduction to a localized subset of the data in a bottom-up fashion.

In this chapter, we propose a new fast Gaussian summation algorithm that enables speedup in higher dimensions. Our approach utilizes: 1) probabilistic relative error bounds (Definition 2.4.3) on kernel sums provided by Monte Carlo estimates; 2) a new tree structure called *subspace tree* for reducing the computational cost of each distance computation. The former can be seen as relaxing the strict requirement of guaranteeing hard relative bound on very small quantities, as done in [82, 77, 80, 114, 112]. The latter was mentioned as a possible way of ameliorating the effects of *the curse of dimensionality* in [137], a pioneering paper in this area. We now formally define the computational task tackled in this chapter.

Problem: Suppose we are given the set of query points $\mathbf{Q}$ and the set of reference points $\mathbf{R}$. Given a pairwise Gaussian kernel function $k(\mathbf{x}, \mathbf{y}) = \exp\left(\frac{-||\mathbf{x}-\mathbf{y}||^2}{2h^2}\right)$, the relative error level $\epsilon > 0$, the probability guarantee level $0 < \alpha < 1$, and the desired kernel sum $\Phi(\mathbf{q}; \mathbf{R}) = \sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{q}, \mathbf{r_j})$ for each $\mathbf{q} \in \mathbf{Q}$,

Task: Compute an approximation $\widetilde{\Phi}(\mathbf{q}; \mathbf{R})$ for each $\mathbf{q} \in \mathbf{Q}$ such that $\left|\widetilde{\Phi}(\mathbf{q}; \mathbf{R}) - \Phi(\mathbf{q}; \mathbf{R})\right| \leq \epsilon \Phi(\mathbf{q}; \mathbf{R})$ with an asymptotic probability guarantee level of α

Algorithm 5.1.1 DFGT($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$)

```
if CanSummarize(Q^sub, R^sub, ε) then
  Summarize(Q^sub, R^sub)
else if CanSummarizeMC(Q^sub, R^sub, ε, α) then
  SummarizeMC(Q^sub, R^sub, ε, α)
else
  if Q^sub is a leaf node then
    if R^sub is a leaf node then
      DFGTBase(Q^sub, R^sub)
    else
      DFGT(Q^sub, R^sub,L), DFGT(Q^sub, R^sub,R)
  else
    if R^sub is a leaf node then
      DFGT(Q^sub,L, R^sub), DFGT(Q^sub,R, R^sub)
    else
      DFGT(Q^sub,L, R^sub,L), DFGT(Q^sub,L, R^sub,R)
      DFGT(Q^sub,R, R^sub,L), DFGT(Q^sub,R, R^sub,R)
```

as fast as possible.

5.1 Gaussian Summation by Monte Carlo Sampling

Here we describe the extension needed for probabilistic computation of kernel summation satisfying Definition 2.4.3. The main routine for the probabilistic kernel summation is shown in Algorithm 5.1.1. The function DFGT takes the query node $\mathbf{Q}^{sub}$ and the reference node $\mathbf{R}^{sub}$ (each initially called with the roots of the query tree and the reference tree). The idea of Monte Carlo sampling used in the new algorithm is similar to the one in [95], except the sampling is done per query and we use approximations that provide hard error bounds as well (i.e. finite difference, exhaustive base case: DFGTBase). This means that the approximation has less variance than a pure Monte Carlo approach used in [95]. Algorithm 5.1.1 first attempts approximations with hard error bounds, which are computationally cheaper than sampling-based approximations. For example, finite-difference scheme [82, 77, 80] can be used for the CanSummarize and Summarize functions in any general dimension.

The CanSummarizeMC function takes two parameters that specify the accuracy: the relative error and its probability guarantee and decides whether to use Monte Carlo sampling for the given pair of nodes. If the reference node $\mathbf{R}^{sub}$ contains too few points, it may be more efficient to process it using exact methods that use error bounds based on bounding primitives on the node pair or exhaustive pair-wise evaluations, which is determined by the condition: $\zeta \cdot m_{initial} \leq |\mathbf{R}^{sub}|$ where $\zeta > 1$ controls the minimum number of reference points needed for Monte Carlo sampling to proceed. If the reference node does contain enough points, then for each query point $\mathbf{q} \in \mathbf{Q}^{sub}$, the Sample routine samples $m_{initial}$ terms over the terms in the summation $\Phi(\mathbf{q}; \mathbf{R}^{sub}) = \sum_{\mathbf{r_{jn}} \in \mathbf{R}^{sub}} k(||\mathbf{q} - \mathbf{r_{jn}}||)$ where $\Phi(\mathbf{q}; \mathbf{R}^{sub})$ denotes the exact contribution of $\mathbf{R}^{sub}$ to $\mathbf{q}$'s kernel sum. Basically, we are interested in estimating $\Phi(\mathbf{q}; \mathbf{R}^{sub})$ by $\widetilde{\Phi}(\mathbf{q}; \mathbf{R}^{sub}) = |\mathbf{R}^{sub}|\mu_{\mathbf{S}}$, where $\mu_{\mathbf{S}}$ is the sample mean of $\mathbf{S}$. From the Central Limit Theorem (Theorem 2.3.2), given enough m samples, $\mu_{\mathbf{S}} \rightsquigarrow \mathcal{N}(\mu, \sigma_S^2/m)$ where $\Phi(\mathbf{q}; \mathbf{R}^{sub}) = |\mathbf{R}^{sub}|\mu$ (i.e. μ is the average of the kernel value between $\mathbf{q}$ and any reference point $\mathbf{r} \in \mathbf{R}^{sub}$); this implies that $|\mu_{\mathbf{S}} - \mu| \leq \frac{z_{\alpha/2}\sigma_{\mathbf{S}}}{\sqrt{m}}$ with probability $1 - \alpha$. The *pruning rule* we have to enforce for each query point for the contribution of $\mathbf{R}^{sub}$ is:

$$z_{\alpha/2}\frac{\sigma_S}{\sqrt{m}} \leq \frac{\epsilon\Phi(\mathbf{q}; \mathbf{R})}{|\mathbf{R}|}$$

where $\sigma_{\mathbf{S}}$ the sample standard deviation of $\mathbf{S}$. Since $\Phi(\mathbf{q}; \mathbf{R})$ is one of the unknown quanities we want to compute, we instead enforce the following:

$$z_{\alpha/2}\frac{\sigma_{\mathbf{S}}}{\sqrt{m}} \leq \frac{\epsilon\left(\Phi^l(\mathbf{q}; \mathbf{R}) + |\mathbf{R}^{sub}|\left(\mu_{\mathbf{S}} - \frac{z_{\alpha/2}\sigma_{\mathbf{S}}}{\sqrt{m}}\right)\right)}{|\mathbf{R}|} \tag{5.1.1}$$

where $\Phi^l(\mathbf{q}; \mathbf{R})$ is the currently running lower bound on the sum computed using exact methods and $|\mathbf{R}^{sub}|\left(\mu_{\mathbf{S}} - \frac{z_{\alpha/2}\sigma_{\mathbf{S}}}{\sqrt{m}}\right)$ is the probabilistic component contributed by $\mathbf{R}^{sub}$. Denoting $\Phi^{l,new}(\mathbf{q}; \mathbf{R}) = \Phi^l(\mathbf{q}; \mathbf{R}) + |\mathbf{R}^{sub}|\left(\mu_{\mathbf{S}} - \frac{z_{\alpha/2}\sigma_{\mathbf{S}}}{\sqrt{|\mathbf{S}|}}\right)$, the **minimum number of samples for $\mathbf{q}$** needed to achieve the target error the right side of the

inequality in Equation 5.1.1 with at least probability of $1-\alpha$ is:

$$m \geq z_{\alpha/2}^2 \sigma_{\mathbf{S}}^2 \frac{(|\mathbf{R}| + \epsilon|\mathbf{R}^{sub}|)^2}{\epsilon^2(\Phi^l(\mathbf{q};\mathbf{R}) + |\mathbf{R}^{sub}|\mu_{\mathbf{S}})^2}$$

If the given query node and reference node pair cannot be pruned using either non-probabilistic/probabilistic approximations, then we recurse on a smaller subsets of two sets. We now state the probablistic error guarantee of our algorithm as a theorem.

Theorem 5.1.1. *After calling* DFGT *with* $\mathbf{Q}^{sub} = \mathbf{Q}$, $\mathbf{R}^{sub} = \mathbf{R}$, *Algorithm 5.1.1 approximates each* $\Phi(\mathbf{q};\mathbf{R})$ *with* $\widetilde{\Phi}(\mathbf{q};\mathbf{R})$ *such that Definition 2.4.3 holds.*

Proof. For a query/reference $(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$ pair and $0 < \alpha < 1$, DFGTBase and Summarize compute estimates for $\mathbf{q} \in \mathbf{Q}^{sub}$ such that $\left|\widetilde{\Phi}(\mathbf{q};\mathbf{R}) - \Phi(\mathbf{q};\mathbf{R})\right| < \epsilon\frac{\Phi(\mathbf{q};\mathbf{R})|\mathbf{R}^{sub}|}{|\mathbf{R}|}$ with probability at least $1-\alpha$. By Equation 5.1.1, SummarizeMC computes estimates for $\mathbf{q} \in \mathbf{Q}^{sub}$ such that $\left|\widetilde{\Phi}(\mathbf{q};\mathbf{R}) - \Phi(\mathbf{q};\mathbf{R})\right| < \epsilon\frac{\Phi(\mathbf{q};\mathbf{R})|\mathbf{R}^{sub}|}{|\mathbf{R}|}$ with probability $1-\alpha$.

We now induct on $|\mathbf{Q}^{sub} \cup \mathbf{R}^{sub}|$. Line 11 of Algorithm 5.1.1 divides over the reference whose subcalls compute estimates that satisfy $\left|\widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub,L}) - \Phi(\mathbf{q};\mathbf{R}^{sub,L})\right| \leq \epsilon\frac{\Phi(\mathbf{q};\mathbf{R})|\mathbf{R}^{sub,L}|}{|\mathbf{R}|}$ and $\left|\widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub,R}) - \Phi(\mathbf{q};\mathbf{R}^{sub,R})\right| \leq \epsilon\frac{\Phi(\mathbf{q};\mathbf{R})|\mathbf{R}^{sub,R}|}{|\mathbf{R}|}$ each with at least $1-\alpha$ probability by induction hypothesis. For $\mathbf{q} \in \mathbf{Q}^{sub}$, $\widetilde{\Phi}(\mathbf{q};\mathbf{R}) = \widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub,L}) + \widetilde{\Phi}(\mathbf{q};\mathbf{R}^{sub,R})$ which means $|\widetilde{\Phi}(\mathbf{q};\mathbf{R}) - \Phi(\mathbf{q};\mathbf{R})| \leq \epsilon\frac{\Phi(\mathbf{q};\mathbf{R})|\mathbf{R}^{sub}|}{|\mathbf{R}|}$ with probability at least $1-\alpha$. Essentially, the frontier nodes used for approximation (see Figure 15) act as strata and error bound argument can be proven using the techniques in Theorem 4 of [95]. Line 14 divides over the query and each subcall computes estimates that hold with at least probability $1-\alpha$ for $\mathbf{q} \in \mathbf{Q}^{sub,L} \cup \mathbf{Q}^{sub,R}$. Line 16 and 17 divides both over the query and the reference, and the correctness can be proven similarly. □

5.2 Subspace Tree

A subspace tree is a space-partitioning tree with a set of orthogonal bases associated with each node $\mathbf{N}$: $\mathbf{\Omega}(N) = (\mu, \mathbf{U}, \mathbf{\Lambda}, d)$ where μ is the mean, $\mathbf{U}$ is a $D \times d$ matrix

Algorithm 5.2.1 Monte Carlo sampling based approximation routines.

SAMPLE($\mathbf{q}, \mathbf{R}^{sub}, \epsilon, \alpha, \mathbf{S}, m$)
for $k = 1$ to m **do**
  $\mathbf{r} \leftarrow$ random point in $\mathbf{R}^{sub}$
  $\mathbf{S} \leftarrow \mathbf{S} \cup \{k(||\mathbf{q} - \mathbf{r}||)\}$
$\mu_{\mathbf{S}} \leftarrow$ MEAN($\mathbf{S}$)
$\sigma^2_{\mathbf{S}} \leftarrow$ VARIANCE($\mathbf{S}$)
$\Phi^{l,new}(\mathbf{q}; \mathbf{R}) \leftarrow \Phi^l(\mathbf{q}; \mathbf{R}) +$
$|\mathbf{R}^{sub}| \left(\mu_{\mathbf{S}} - \frac{z_{\alpha/2}\sigma_{\mathbf{S}}}{\sqrt{|\mathbf{S}|}} \right)$
$m_{thresh} \leftarrow z^2_{\alpha/2}\sigma^2_{\mathbf{S}} \frac{(|\mathbf{R}|+\epsilon|\mathbf{R}^{sub}|)^2}{\epsilon^2(\Phi^l(\mathbf{q};\mathbf{R})+|\mathbf{R}^{sub}|\mu_{\mathbf{S}})^2}$
$m \leftarrow m_{thresh} - |\mathbf{S}|$

CANSUMMARIZEMC($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \epsilon, \alpha$)
return $\zeta \cdot m_{initial} \leq |\mathbf{R}^{sub}|$

SUMMARIZEMC($\mathbf{Q}^{sub}, \mathbf{R}^{sub}, \epsilon, \alpha$)
for $\mathbf{q_i} \in \mathbf{Q}^{sub}$ **do**
  $\mathbf{S} \leftarrow \emptyset$, $m \leftarrow m_{initial}$
  repeat
    SAMPLE($\mathbf{q_i}, \mathbf{R}^{sub}, \epsilon, \alpha, \mathbf{S}, m$)
  until $m \leq 0$
  $\Phi(\mathbf{q_i}; \mathbf{R}) \leftarrow \Phi(\mathbf{q_i}; \mathbf{R}) + |\mathbf{R}^{sub}| \cdot$
  MEAN($\mathbf{S}$)

whose columns consist of d eigenvectors, and $\mathbf{\Lambda}$ the corresponding eigenvalues. The orthogonal basis set is constructed using a linear dimension reduction method such as PCA. It is constructed in the top-down manner using the PARTITIONSET function dividing the given set of points into two (where the PARTITIONSET function divides along the dimension with the highest variance in case of a *kd*-tree for example), with the subspace in each node formed in the bottom-up manner. Algorithm 5.3.1 shows a PCA tree (a subspace tree using PCA as a dimension reduction) for a 3-D dataset. The subspace of each leaf node is computed using PCABASE which can use the exact PCA [76] or a stochastic one [60]. For an internal node, the subspaces of the child nodes, $\mathbf{\Omega}(\mathbf{N}^L) = (\mu_{\mathbf{L}}, \mathbf{U_L}, \mathbf{\Lambda_L}, d_L)$ and $\mathbf{\Omega}(\mathbf{N}^R) = (\mu_{\mathbf{R}}, \mathbf{U_R}, \mathbf{\Lambda_R}, d_R)$, are approximately merged using the MERGESUBSPACES function which involves solving an $(d_L + d_R + 1) \times (d_L + d_R + 1)$ eigenvalue problem [90], which runs in $\mathcal{O}((d_L + d_R + 1)^3) \ll \mathcal{O}(D^3)$ given that the dataset is sparse. In addition, each data point $\mathbf{x}$ in each node $\mathbf{N}$ is mapped to its new lower-dimensional coordinate using the orthogonal basis set of $\mathbf{N}$: $\mathbf{x}_{proj} = \mathbf{U}^T(\mathbf{x} - \mu)$. The L_2 norm reconstruction error is given by: $||\mathbf{x}_{recon} - \mathbf{x}||_2^2 = ||(\mathbf{U}\mathbf{x}_{proj} + \mu) - \mathbf{x}||_2^2$.

Monte Carlo Sampling using a Subspace Tree. Consider CANSUMMARIZEMC function in Algorithm 5.2.1. The "outer-loop" over this algorithm is over the query set $\mathbf{Q}$, and it would make sense to project each query point $\mathbf{q} \in \mathbf{Q}$ to the subspace

owned by the reference node $\mathbf{R}^{sub}$. Let $\mathbf{U}$ and μ be the orthogonal basis system for $\mathbf{R}^{sub}$ consisting of d basis. For each $\mathbf{q} \in \mathbf{Q}^{sub}$, consider the squared distance $||(\mathbf{q}-\mu)-\mathbf{r}_{proj}||^2$ (where $(\mathbf{q}-\mu)$ is $\mathbf{q}$'s coordinates expressed in terms of the coordinate system of $\mathbf{R}^{sub}$) as shown in Figure 32. For the Gaussian kernel, each pairwise kernel value is approximated as:

$$\exp\left(\frac{-||\mathbf{q}-\mathbf{r}||^2}{2h^2}\right) \approx \exp\left(\frac{-||\mathbf{q}-\mathbf{q}_{recon}||^2}{2h^2}\right)\exp\left(\frac{-||\mathbf{q}_{proj}-\mathbf{r}_{proj}||^2}{2h^2}\right) \qquad (5.2.1)$$

where $\mathbf{q}_{recon} = \mathbf{U}\mathbf{q}_{proj} + \mu$ and $\mathbf{q}_{proj} = \mathbf{U}^T(\mathbf{q}-\mu)$. For a fixed query point $\mathbf{q}$, $\exp\left(\frac{-||\mathbf{q}-\mathbf{q}_{recon}||^2}{2h^2}\right)$ can be precomputed (which takes d dot products between two D-dimensional vectors) and re-used for every distance computation between $\mathbf{q}$ and any reference point $\mathbf{r} \in \mathbf{R}^{sub}$ whose cost is now $\mathcal{O}(d) \ll \mathcal{O}(D)$. Therefore, we can take more samples efficiently. For a total of sufficiently large m samples, the computational cost is $\mathcal{O}(d(D+m)) \ll \mathcal{O}(D \cdot m)$ for each *query point*.

Increased variance comes at the cost of inexact distance computations, however. Each distance computation incurs at most squared L_2 norm of $||\mathbf{r}_{recon} - \mathbf{r}||_2^2$ error. That is, $|||\mathbf{q}-\mathbf{r}_{recon}||_2^2 - ||\mathbf{q}-\mathbf{r}||_2^2| \leq ||\mathbf{r}_{recon}-\mathbf{r}||_2^2$. Neverhteless, the sample variance for each query point **plus** the inexactness due to dimension reduction ζ_S can be shown to be bounded for the Gaussian kernel as: (where each $s = \exp\left(\frac{-||\mathbf{q}-\mathbf{r}_{recon}||^2}{2h^2}\right)$):

$$\frac{1}{m-1}\left(\sum_{s\in\mathbf{S}} s^2 - m\cdot\mu_{\mathbf{S}}^2\right) + \zeta_{\mathbf{S}}$$
$$\leq \frac{1}{m-1}\left(\left(\sum_{s\in\mathbf{S}} s^2\right)\min\left\{1, \max_{\mathbf{r}\in\mathbf{R}^{sub}} \exp\left(\frac{||\mathbf{r}_{recon}-\mathbf{r}||_2^2}{h^2}\right)\right\} - m\left(\mu_{\mathbf{S}} \min_{\mathbf{r}\in\mathbf{R}^{sub}} \exp\left(\frac{-||\mathbf{r}_{recon}-\mathbf{r}||_2^2}{2h^2}\right)\right)^2\right)$$

Exhaustive Computations Using a Subspace Tree. Now suppose we have built subspace trees for the query and the reference sets. We can project either each query point onto the reference subspace, or each reference point onto the query subspace, depending on which subspace has a smaller dimension and the number of points in each node. The subspaces formed in the leaf nodes usually are highly numerically accurate since it contains very few points.

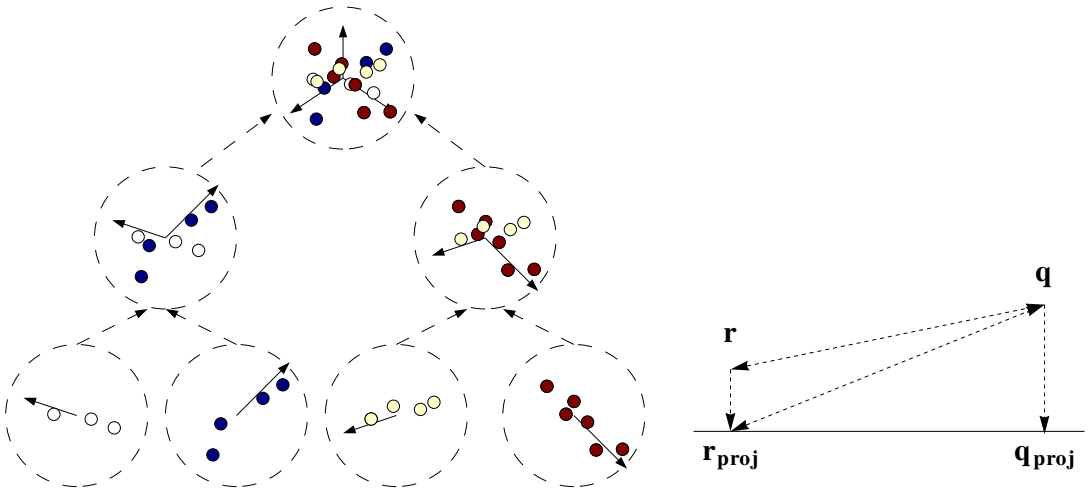


Figure 32: Left: A PCA-tree for a 3-D dataset. Note that the tree is constructed in the original Euclidean space that contains the point set, but instead here shown to illustrate how subspaces are merged. Right: The squared distance between a given query point and a reference point projected onto a subspace can be decomposed into the orthogonal component and the subspace component.

5.3 Experimental Results

We empirically evaluated the runtime performance of our algorithm on seven real-world datasets, scaled to fit in $[0, 1]^D$ hypercube, for approximating the Gaussian sum at every query point with a range of bandwidths. This experiment is motivated by many kernel methods that require computing the Gaussian sum at different bandwidth values (according to the standard least-sqares cross-validation scores [170]). Nevertheless, we emphasize that the acceleration results are applicable to other kernel methods that require efficient Gaussian summation.

In this chapter, the reference set equals the query set. All datasets have 50K points so that the exact exhaustive method can be tractably computed. All times are in seconds and include the time needed to build the trees. Codes are in C/C++ and run on a dual Intel Xeon 3GHz with 8 Gb of main memory. The measurements in second to eigth columns are obtained by running the algorithms at the bandwidth kh^* where $10^{-3} \leq k \leq 10^3$ is the constant in the corresponding column header. The last columns denote the total time needed to run on all seven bandwidth values.

Algorithm 5.3.1 PCA tree building routine.

```
BuildPcaTree(P^sub)
if CanPartition(P^sub) then
  {P^sub,L, P^sub,R} ← PartitionSet(P^sub)
  N ← empty node
  N^L ← BuildPcaTree(P^L)
  N^R ← BuildPcaTree(P^R)
  MergeSubspaces(Subspace of N^L, Subspace of N^R)
else
  N ← BuildPcaTreeBase(P^sub)
  PcaBase(P^sub)
Project(P^sub, Subspace of N)
return N
```

Each table has results for five algorithms: the naive algorithm and four algorithms. The algorithms with $\alpha = 0$ denote the previous state-of-the-art (finite-difference with error redistribution) [112], while those with $\alpha > 0$ denote our probabilistic version. Each entry has the running time and the percentage of the query points that **did not** satisfy the relative error ϵ.

Analysis. Readers should focus on the last columns containing the total time needed for evaluating Gaussian sum at all points for seven different bandwidth values. This is indicated by boldfaced numbers for our probabilistic algorithm. As expected, On low-dimensional datasets (below 6 dimensions), the algorithm using series-expansion based bounds gives two to three times speedup compared to our approach that uses Monte Carlo sampling. Multipole moments are an effective form of compression in low dimensions with analytical error bounds that can be evaluated; our Monte Carlo-based method has an asymptotic error bound which must be "learned" through sampling. As we go from 7 dimensions and beyond, series-expansion cannot be done efficiently because of its slow convergence. Our probabilistic algorithm ($\alpha = 0.9$) using Monte Carlo consistently performs better than the algorithm using exact bounds ($\alpha = 0$) by at least a factor of two. Compared to naive, it achieves the maximum speedup of about nine times on an 16-dimensional dataset; on an 89-dimensional dataset, it is

at least three times as fast as the naive. Note that all the datasets contain only 50K points, and the speedup will be more dramatic as we increase the number of points.

5.4 Conclusion and Future Work

We presented an extension to fast multipole methods to use approximation methods with both hard and probabilistic bounds. Our new technique is based on a probabilistic approximation based on the central limit theorem and a new data structure that records a dominant subspace in each node in a hierarchical data structure which reduces the distance computation cost. Our experimental results show speedup over the previous state-of-the-art on high-dimensional datasets. Our future work will include possible improvements inspired by a recent work done in the FMM community using a matrix-factorization formulation [129] and a more extensive experimental comparison.

Algorithm \ scale	*0.001*	*0.01*	*0.1*	*1*	*10*	*100*	*1000*	Σ
mockgalaxy-D-1M-rnd (cosmology: positions), $D = 3, N = 50000, h^* = 0.000768201$								
Naive	182	182	182	182	182	182	182	1274
MCMM	3	3	5	10	26	48	2	**97**
$(\epsilon = 0.1, \alpha = 0.1)$	1 %	1 %	1 %	1 %	1 %	1 %	5 %	
DFGT	2	2	2	2	6	19	3	36
$(\epsilon = 0.1, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
MCMM	3	3	4	11	27	58	21	**127**
$(\epsilon = 0.01, \alpha = 0.1)$	0 %	0 %	1 %	1 %	1 %	1 %	7 %	
DFGT	2	2	2	2	7	30	5	50
$(\epsilon = 0.01, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
bio5-rnd (biology: drug activity), $D = 5, N = 50000, h^* = 0.000567161$								
Naive	214	214	214	214	214	214	214	1498
MCMM	4	4	6	144	149	65	1	**373**
$(\epsilon = 0.1, \alpha = 0.1)$	0 %	0 %	0 %	0 %	1 %	0 %	1 %	
DFGT	4	4	5	24	96	65	2	200
$(\epsilon = 0.1, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
MCMM	4	4	6	148	165	126	1	**454**
$(\epsilon = 0.01, \alpha = 0.1)$	0 %	0 %	0 %	0 %	1 %	0 %	1 %	
DFGT	4	4	5	25	139	126	4	307
$(\epsilon = 0.01, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
$pall7 - rnd, D = 7, N = 50000, h^* = 0.00131865$								
Naive	327	327	327	327	327	327	327	2289
MCMM	3	3	3	3	63	224	< 1	**300**
$(\epsilon = 0.1, \alpha = 0.1)$	0 %	0 %	0 %	1 %	1 %	12 %	0 %	
DFGT	10	10	11	14	84	263	223	615
$(\epsilon = 0.1, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
MCMM	3	3	3	3	70	265	5	**352**
$(\epsilon = 0.01, \alpha = 0.1)$	0 %	0 %	0 %	1 %	2 %	1 %	8 %	
DFGT	10	10	11	14	85	299	374	803
$(\epsilon = 0.01, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
$covtype - rnd, D = 10, N = 50000, h^* = 0.0154758$								
Naive	380	380	380	380	380	380	380	2660
MCMM	11	11	13	39	318	< 1	< 1	**381**
$(\epsilon = 0.1, \alpha = 0.1)$	0 %	0 %	0 %	1 %	0 %	0 %	0 %	
DFGT	26	27	38	177	390	244	< 1	903
$(\epsilon = 0.1, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
MCMM	11	11	13	77	362	2	< 1	**477**
$(\epsilon = 0.01, \alpha = 0.1)$	0 %	0 %	0 %	1 %	1 %	10 %	0 %	
DFGT	26	27	38	180	427	416	< 1	1115
$(\epsilon = 0.01, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	

Algorithm \ scale	*0.001*	*0.01*	*0.1*	*1*	*10*	*100*	*1000*	Σ
$CoocTexture - rnd, D = 16, N = 50000, h^* = 0.0263958$								
Naive	472	472	472	472	472	472	472	3304
MCMM	10	11	22	189	109	< 1	< 1	**343**
$(\epsilon = 0.1, \alpha = 0.1)$	0 %	0 %	0 %	1 %	8 %	0 %	0 %	
DFGT	22	26	82	240	452	66	< 1	889
$(\epsilon = 0.1, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
MCMM	10	11	22	204	285	< 1	< 1	**534**
$(\epsilon = 0.01, \alpha = 0.1)$	0 %	0 %	1 %	1 %	10 %	4 %	0 %	
DFGT	22	26	83	254	543	230	< 1	1159
$(\epsilon = 0.01, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
$LayoutHistogram - rnd, D = 32, N = 50000, h^* = 0.0609892$								
Naive	757	757	757	757	757	757	757	5299
MCMM	32	32	54	168	583	8	8	**885**
$(\epsilon = 0.1, \alpha = 0.1)$	0 %	0 %	1 %	1 %	1 %	0 %	0 %	
DFGT	153	159	221	492	849	212	< 1	2087
$(\epsilon = 0.1, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
MCMM	32	45	60	183	858	8	8	**1246**
$(\epsilon = 0.01, \alpha = 0.1)$	0 %	0 %	1 %	6 %	1 %	0 %	0 %	
DFGT	153	159	222	503	888	659	< 1	2585
$(\epsilon = 0.01, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
$CorelCombined - rnd, D = 89, N = 50000, h^* = 0.0512583$								
Naive	1716	1716	1716	1716	1716	1716	1716	12012
MCMM	384	418	575	428	1679	17	17	**3518**
$(\epsilon = 0.1, \alpha = 0.1)$	0 %	0 %	0 %	1 %	10 %	0 %	0 %	
DFGT	659	677	864	1397	1772	836	17	6205
$(\epsilon = 0.1, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	
MCMM	401	419	575	437	1905	17	17	**3771**
$(\epsilon = 0.01, \alpha = 0.1)$	0 %	0 %	0 %	1 %	2 %	0 %	0 %	
DFGT	659	677	865	1425	1794	1649	17	7086
$(\epsilon = 0.01, \alpha = 0)$	0 %	0 %	0 %	0 %	0 %	0 %	0 %	

CHAPTER VI

APPLICATIONS IN NONPARAMETRIC CLUSTERING

Mean shift is a powerful but computationally expensive method for nonparametric clustering and optimization. It iteratively moves each data point to its local mean until convergence. We introduce a fast algorithm for computing mean shift based on the dual-tree. Unlike previous speed-up attempts, our algorithm maintains a relative error bound at each iteration, resulting in significantly more stable and accurate convergence. We demonstrate the benefit of our method in clustering experiments with real and synthetic data.

This chapter presents a fast algorithm for computing mean shift (MS). MS is a nonparametric, iterative method for unsupervised clustering and global/local optimization. It has a wide range of applications in clustering and data analysis. For example, the computer vision community has utilized MS for (1) its clustering property in image segmentation, feature analysis [48] and texture classification [73]; and for (2) its quadratic optimization property in visual tracking [46, 47]. MS is attractive for clustering and optimization problems due to its ability to adapt to the data distribution. However, it suffers from high computational cost - $\mathcal{O}(|\mathbf{Q}||\mathbf{R}|)$ operations in each iteration (see the pseudo code in algorithm 6.1.1) Therefore, applications of MS have either used fairly small datasets [48, 47], or avoided updating all of the points in the query set (e.g. a local optimization process is started from a single query). Alternatively, some fast approximations of MS have been proposed [73, 201]. While these methods have been shown experimentally to have high efficiency, they suffer from three major limitations: 1) Improved Fast Gauss Transform-based MS [201] (IFGT-MS) can use only the Gaussian kernel; 2) Both IFGT-MS and Locality Sensitive

Hashing-based MS [73] (LSH-MS) have many tuning parameters; 3) Both methods lack explicit error bounds for the vector approximation in each iteration of MS.

We believe speedup techniques should ensure both the *accuracy* and the *stability* of the approximation. "Accuracy" means that the approximation has a guaranteed error bound. "Stability" means that the approximation should return almost identical results over different runs. Nondeterminism typically stems from randomized initialization, and approximation methods which lack reliable error control mechanisms can be sensitive to these initial values, resulting in a significant variation in their outputs for a fixed input. In this chapter, we introduce an acceleration technique that achieves both accuracy and stability – Dual-tree [78] based mean shift (DT-MS). DT-MS can use any kernel, has a user-specified *relative* error tolerance on each computation of $\mathbf{m}(\mathbf{q_i})$ (Equation (6.1.1)) and requires no other parameter tuning. Our experiments on datasets with dimensionality ranging from 2 to 16 and size ranging from $6,000$ to $68,040$ demonstrate the superiority of DT-MS over IFGT-MS and LSH-MS in terms of speed, accuracy, and stability. This chapter makes three contributions:

1. Introduction of DT-MS, a novel approximation method for MS which is fast, accurate, and stable.

2. An extension of the dual-tree method (introduced in [78] for positive scalar targets) to the signed mean vector case. To achieve this extension, we have developed (i) A new global error bound (Theorem 1) for pruning nodes, (ii) A novel finite difference approximation for the signed mean vector, and (iii) A new algorithm for updating bounds on the L1 norm.

3. The first empirical comparison of fast MS algorithms on standardized datasets. We highlight for the first time the issue of stability in MS approximation.

6.1 Mean Shift

Algorithm 6.1.1 MEAN-SHIFT($\mathbf{Q}, \mathbf{R}, w(\cdot), k(\cdot), \epsilon$)

Input: $\mathbf{Q}$, $\mathbf{R}$, ϵ (the pre-defined distance threshold)
Output: The converged query set

$dist = 100 * ones(|\mathbf{Q}|, 1)$ {initialize distance vector}
while max(dist)$\geq \epsilon$ **do**
  for each $\mathbf{q_i} \in \mathbf{Q}$ **do**
    $\mathbf{m}(\mathbf{q_i}) = \frac{\sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{r_j} - \mathbf{q_i}) w(\mathbf{r_j}) \mathbf{r_j}}{\sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{r_j} - \mathbf{q_i}) w(\mathbf{r_j})}$
    $dist(\mathbf{q_i}) = \|\mathbf{m}(\mathbf{q_i}) - \mathbf{q_i}\|_2$ {distance can be of any norm}
    $\mathbf{q_i} \leftarrow \mathbf{m}(\mathbf{q_i})$
return $\mathbf{Q}$

Mean shift [38, 71] moves each query to its local mean until convergence (see algorithm 6.1.1). Let $\mathbf{R}$ denote the reference data set, and $\mathbf{Q}$ denote the query data set. $\mathbf{R} \subset \mathbb{R}^D, \mathbf{Q} \subset \mathbb{R}^D, \mathbf{r_j} \in \mathbf{R}, \mathbf{q_i} \in \mathbf{Q}$. The mean of query $\mathbf{q_i}$ is defined as:

$$\mathbf{m}(\mathbf{q_i}) = \frac{\mathbf{h}(\mathbf{q_i})}{f(\mathbf{q_i})} = \frac{\sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{r_j} - \mathbf{q_i}) w(\mathbf{r_j}) \mathbf{r_j}}{\sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{r_j} - \mathbf{q_i}) w(\mathbf{r_j})} \tag{6.1.1}$$

where $w : \mathbb{R}^D \rightarrow \mathbb{R}$ is a weight function which can vary with $\mathbf{r_j}$ and time. In this chapter we set $w(\mathbf{r_j}) = 1$ for all $\mathbf{r_j}$. The kernel function $k : \mathbb{R}^D \rightarrow \mathbb{R}$ has profile $k : [0, \infty] \rightarrow R$, such that $k(x) = k(\|\frac{x}{h}\|^2)$, where h is the bandwidth and k is monotonically non-increasing, nonnegative and piecewise continuous [38]. Now we formally define the computational task tackled in this chapter[1].

Problem: Suppose we are given the set of query points $\mathbf{Q}$ and the set of reference points $\mathbf{R}$ in each mean shift iteration. Given a pairwise kernel function k, the relative error level $\epsilon > 0$, and the desired kernel sums $\mathbf{h}(\mathbf{q_i}) = \sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{r_j} - \mathbf{q_i}) w(\mathbf{r_j}) \mathbf{r_j}$ and $f(\mathbf{q_i}) = \sum_{\mathbf{r_j} \in \mathbf{R}} k(\mathbf{r_j} - \mathbf{q_i}) w(\mathbf{r_j})$,

Task: Compute an approximation $\widetilde{\mathbf{m}}(\mathbf{q}; \mathbf{R}) = \frac{\widetilde{\mathbf{h}}(\mathbf{q_i})}{\widetilde{f}(\mathbf{q_i})}$ for each $\mathbf{q} \in \mathbf{Q}$ such that

[1] We realize that this is speeding up only the inner-loop computations in the mean shift clustering procedure. The error accumulation incurred across multiple iterations do affect the convergence but we experimentally validate that the resulting clusters are more stable than previous approaches.

$||\widetilde{\mathbf{m}}(\mathbf{q};\mathbf{R}) - \mathbf{m}(\mathbf{q};\mathbf{R})|\,|_1 \leq \epsilon||\mathbf{m}(\mathbf{q};\mathbf{R})||_1$ as fast as possible for speeding up each iteration of the mean shift clustering procedure (i.e. bounding the L_1 norm error).

Cheng [38] proves that MS is a step-varying gradient ascent optimization. [65] shows MS is equivalent to Newton's method with piecewise constant kernels, and it is a quadratic bound maximization for all kernels.

6.2 Previous Acceleration Methods

The denominator of Equation 6.1.1 is a kernel density estimate (KDE) while the numerator is a weighted vector sum. The key challenge in accelerating MS is to approximate this ratio. Since MS is closely related to KDE, most speedup methods focus on fast approximation of $f(\mathbf{q_i})$ or fast range search at $\mathbf{q_i}$ (defined by the bandwidth). The two most important related works are the Improved Fast Gauss Transform-based MS (IFGT-MS) [201] and Locality Sensitive Hashing-based MS (LSH-MS) [73].

IFGT-MS is applicable to only the Gaussian kernel. IFGT-MS first clusters the reference points using the k-center algorithm and loops over each query point/reference cluster pair, evaluating the precomputed (truncated) Taylor coefficients for clusters that are within the distance threshold from the query point. IFGT-MS requires a significant amount of manual parameter tuning for good performance.[2]

LSH [74] has been popular recently for k-nearest neighbor (k-NN) search in high dimensions. It performs L random partitions of the data set. For each partition, a boolean vector of size K is generated for each datum, thus the data set are indexed into 2^K cells. Each query $\mathbf{q_i}$ belongs to L cells simultaneously. The union of the L cells is returned as the neighborhood of $\mathbf{q_i}$. The choice of (K, L) is critical. The training process [73] selects the (K, L) that minimizes the query time on a subset

[2]The important parameters are: p-polynomial order, K_c-number of partitions, e-ratio of the cutoff radius to the bandwidth, which determines the absolute error bound. We follow the authors' suggestion: $K_c = \sqrt{|\mathbf{R}|}$; we gradually increase e and p as we tune them to achieve comparable result to DT-MS(Epan.), though the authors recommend $e = 3$ and $p \leq 3$.

of $\mathbf{Q}$ and satisfies a user-specified k-NN distance approximation error bound, which unfortunately is not directly related to the approximation of $\mathbf{m}(\mathbf{q_i})$.

Unlike these two previous speedup techniques, our dual-tree based mean shift method imposes a relative error bound on the entire mean vector $\mathbf{m}(\mathbf{q_i})$. Achieving this stronger accuracy guarantee requires more computation than other approaches, but our experimental results demonstrate that DT-MS achieves much more stable convergence results while still providing a significant speedup. In particular, DT-MS is faster than IFGT-MS, LSH-MS and naive MS in speed and convergence when using the Epanechnikov kernel.

6.3 Dual-tree Mean Shift

The dual-tree-based algorithm described in Chapter 2 and Chapter 3 can be applied to the mean shift computation because it computes $\mathbf{m}(\mathbf{q_i}) = \mathbf{h}(\mathbf{q_i})/f(\mathbf{q_i})$ (which involves summations of weighted pairwise kernel values) in every iteration of MS. In every iteration of MS, a query tree is rebuilt because $\mathbf{q_i}$ is updated as $\mathbf{m}(\mathbf{q_i})$, while the reference tree remains fixed. In contrast to KDE, mean shift involves the numerator $\mathbf{h}(\mathbf{q_i})$ which is a weighted vector sum and $f(\mathbf{q_i})$ which is in the form of KDE. Here we ensure a relative error bound in L_1 norm (other norms are applicable too) on the mean vector $\mathbf{m}(\mathbf{q_i})$ directly: $|\widetilde{\mathbf{h}}(\mathbf{q_i})/\widetilde{f}(\mathbf{q_i}) - \mathbf{h}(\mathbf{q_i})/f(\mathbf{q_i})|_1/|\mathbf{h}(\mathbf{q_i})/f(\mathbf{q_i})|_1 \leq \tau$. $\widetilde{\mathbf{h}}(\mathbf{q_i})$ and $\widetilde{f}(\mathbf{q_i})$ denote approximations to the numerator and the denominator, respectively.

This error bound brings up three questions: 1) How to distribute the global error bound τ into the local node-node pruning? 2) How to maintain the bounds for the vector? 3) How to apply these bounds in approximation? We answer these below.

Maintaining the Bounds. The distance bounds between $\mathbf{Q}^{sub}$ and $\mathbf{R}^{sub}$, and hence the bounds on $f(\mathbf{q_i})$ and $\mathbf{h}(\mathbf{q_i})$, are used in the linear approximation and error bounds distribution. Unlike KDE, the vector term takes on both positive and negative values, so we need to keep track of them separately. For each query point $\mathbf{q_i}$ and for each query

Algorithm 6.3.1 MS-DUALTREE($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$)

if $\mathbf{Q}^{sub}$ is not a leaf node **then**
  for each dimension d **do**
    $h^{l,d}(\mathbf{q_i}) = \max\left(h^{l,d}(\mathbf{q_i}), \min\left(h^{min}_{\mathbf{Q}^{sub,L},d}, h^{min}_{\mathbf{Q}^{sub,R},d}\right)\right)$
    $h^{u,d}(\mathbf{q_i}) = \min\left(h^{u,d}(\mathbf{q_i}), \max\left(h^{max}_{\mathbf{Q}^{sub,L},d}, h^{max}_{\mathbf{Q}^{sub,R},d}\right)\right)$
  $f^{l}(\mathbf{Q}^{sub}) = \max\left(f^{l}(\mathbf{Q}^{sub}), \min\left(f^{min}_{\mathbf{Q}^{sub,L}}, f^{min}_{\mathbf{Q}^{sub,R}}\right)\right)$
  $f^{u}(\mathbf{q_i}) = \min\left(f^{u}(\mathbf{q_i}), \max\left(f^{max}_{\mathbf{Q}^{sub,L}}, f^{max}_{\mathbf{Q}^{sub,R}}\right)\right)$
$\Delta K = k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub}))$
$\Delta K_{min} = k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - k(d^l(\mathbf{Q}, \mathbf{R}))$
$\Delta K_{max} = k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - k(d^u(\mathbf{Q}, \mathbf{R}))$
$dl_f = |\mathbf{R}^{sub}|\Delta K_{max}$, $du_f = |\mathbf{R}^{sub}|\Delta K_{min}$.
if $\Delta K \leq \min\left\{\frac{\tau f^l(\mathbf{Q}^{sub})L(\mathbf{Q}^{sub})}{NU(\mathbf{Q}^{sub})}, \frac{\tau L(\mathbf{Q}^{sub})}{\sum_d S^{A,d}}\right\}$ **then**
  for each dimension d **do**
    $dl_{h_d} = S^{-,d}(\mathbf{R}^{sub})\Delta K_{min} + S^{+,d}(\mathbf{R}^{sub})\Delta K_{max}$
    $du_{h_d} = S^{+,d}(\mathbf{R}^{sub})\Delta K_{min} + S^{-,d}(\mathbf{R}^{sub})\Delta K_{max}$
    $h^{l,d}(\mathbf{q_i}) += dl_{h_d}$, $h^{u,d}(\mathbf{q_i}) += du_{h_d}$
  $f^{l}(\mathbf{Q}^{sub}) += dl_f$, $f^{u}(\mathbf{Q}^{sub}) += du_f$
else if $\mathbf{Q}^{sub}$ and $\mathbf{R}^{sub}$ are leaves **then**
  MS-DUALTREEBASE($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$)
else
  MS-DUALTREE($\mathbf{Q}^{sub,L}, \mathbf{R}^{sub,L}$), MS-DUALTREE($\mathbf{Q}^{sub,L}, \mathbf{R}^{sub,R}$)
  MS-DUALTREE($\mathbf{Q}^{sub,R}, \mathbf{R}^{sub,L}$), MS-DUALTREE($\mathbf{Q}^{sub,R}, \mathbf{R}^{sub,R}$)

node $\mathbf{Q}^{sub}$, we maintain dimension-wise lower and upper bounds for the numerator: $h^{l,d}(\mathbf{q_i})$ and $h^{u,d}(\mathbf{q_i})$ for $\mathbf{q_i}$, and $h^{l,d}(\mathbf{Q}^{sub})$ and $h^{u,d}(\mathbf{Q}^{sub})$ for $\mathbf{Q}^{sub}$ for $1 \leq d \leq D$, denoted $h^l(\mathbf{q_i})$, $h^u(\mathbf{q_i})$, $h^l(\mathbf{Q}^{sub})$, and $h^u(\mathbf{Q}^{sub})$ collectively as a vector. Similarly, the lower and the upper bounds for the denominator can be maintained: $f^l(\mathbf{q_i})$, $f^u(\mathbf{q_i})$, $f^l(\mathbf{Q}^{sub})$, and $f^u(\mathbf{Q}^{sub})$. We define the following sums of directional coordinate values for all reference points: $S^{A,d} = \sum_{\mathbf{r_j} \in \mathbf{R}} |\mathbf{r_j}(d)|$, $S^{+,d} = \sum_{\mathbf{r_j} \in \mathbf{R}, \mathbf{r_j}(d) > 0} \mathbf{r_j}(d)$, $S^{-,d} = \sum_{\mathbf{r_j} \in \mathbf{R}, \mathbf{r_j}(d) < 0} \mathbf{r_j}(d)$, and the sums for reference points belonging to a given reference node $\mathbf{R}^{sub}$: $S^{+,d}(\mathbf{R}^{sub}) = \sum_{\mathbf{r_j} \in \mathbf{R}^{sub}, \mathbf{r_j}(d) > 0} \mathbf{r_j}(d)$, $S^{-,d}(\mathbf{R}^{sub}) = \sum_{\mathbf{r_j} \in \mathbf{R}^{sub}, \mathbf{r_j}(d) < 0} \mathbf{r_j}(d)$, $S^{d}(\mathbf{R}^{sub}) = S^{+,d}(\mathbf{R}^{sub}) + S^{-,d}(\mathbf{R}^{sub})$, $S^{A,d}(\mathbf{R}^{sub}) = \sum_{\mathbf{r_j} \in \mathbf{R}^{sub}} |\mathbf{r_j}(d)|$, where $\mathbf{r_j}(d)$ is the d^{th} coordinate of $\mathbf{r_j}$, $d = 1, ..., D$. After the query and the reference trees are built, we initialize the lower and the upper bounds for the numerator and the denominator for all $\mathbf{q_i}$'s and all $\mathbf{Q}^{sub}$'s as follows:

Algorithm 6.3.2 MS-DUALTREEBASE($\mathbf{Q}^{sub}, \mathbf{R}^{sub}$)

for each $\mathbf{q_i} \in \mathbf{Q}^{sub}$ **do**
  for each $\mathbf{r_j} \in \mathbf{R}^{sub}$ **do**
    $c = k(\|\mathbf{q_i} - \mathbf{r_j}\|)$, $f_q^{min} += c$, $f_q^{max} += c$, $f_q^{min} -= |\mathbf{R}^{sub}|k(d^u(\mathbf{Q}, \mathbf{R}))$, $f_q^{max} -= |\mathbf{R}^{sub}|k(d^l(\mathbf{Q}, \mathbf{R}))$
    for each dimension d **do**
      $h^{l,d}(\mathbf{q_i}) += c \cdot \mathbf{r_j}(d)$,$h^{u,d}(\mathbf{q_i}) += c \cdot \mathbf{r_j}(d)$,
      $h^{l,d}(\mathbf{q_i}) -= (S_{R,d}^{-}k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) + S_{R,d}^{+}k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})))$,
      $h^{u,d}(\mathbf{q_i}) -= (S_{R,d}^{-}k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) + S_{R,d}^{+}k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})))$
$f^l(\mathbf{Q}^{sub}) = \min_{q \in Q} f_q^{min}$, $f_Q^{max} = \max_{q \in Q} f_q^{max}$
for each dimension d **do**
  $h^{l,d}(\mathbf{q_i}) = \min_{q \in Q} h^{l,d}(\mathbf{q_i})$, $h^{u,d}(\mathbf{q_i}) = \max_{q \in Q} h^{u,d}(\mathbf{q_i})$

$$h^{l,d}(\mathbf{q_i}) = h^{l,d}(\mathbf{Q}^{sub}) = S^{-,d}k(d^l(\mathbf{Q}, \mathbf{R})) + S^{+,d}k(d^u(\mathbf{Q}, \mathbf{R}))$$

$$h^{u,d}(\mathbf{q_i}) = h^{u,d}(\mathbf{Q}^{sub}) = S^{+,d}k(d^l(\mathbf{Q}, \mathbf{R})) + S^{-,d}k(d^u(\mathbf{Q}, \mathbf{R}))$$

$$f^l(\mathbf{q_i}) = f^l(\mathbf{Q}^{sub}) = |\mathbf{R}|k(d^u(\mathbf{Q}, \mathbf{R}))$$

$$f^u(\mathbf{q_i}) = f^u(\mathbf{Q}^{sub}) = |\mathbf{R}|k(d^l(\mathbf{Q}, \mathbf{R}))$$

where $d^l(\mathbf{Q}, \mathbf{R})$ and $d^u(\mathbf{Q}, \mathbf{R})$ denote the min/max distances between the root node of the query tree and the root node of the reference tree. The bounds above will be maintained and updated at all times, such that for any query node $\mathbf{Q}^{sub}$: $h^{l,d}(\mathbf{q_i}) \leq h^d(\mathbf{q_i}) \leq h^{u,d}(\mathbf{q_i})$ for $1 \leq d \leq D$ and $f^l(\mathbf{Q}^{sub}) \leq f(\mathbf{q_i}) \leq f^u(\mathbf{Q}^{sub})$ for any $\mathbf{q_i} \in \mathbf{Q}^{sub}$.

Specifying the Summarize Function. Given a query node $\mathbf{Q}^{sub}$ and a reference node $\mathbf{R}^{sub}$, we can approximate $\mathbf{R}^{sub}$'s contribution to the numerator as $\mathbf{h}(\mathbf{q_i}; \mathbf{R}^{sub})$ and to the denominator as $f(\mathbf{q_i}; \mathbf{R}^{sub})$ for all $\mathbf{q_i} \in \mathbf{Q}^{sub}$ by the linear finite difference approximation with the bounds for $d = 1, \cdots, D$:

$$\widetilde{h}^d(\mathbf{q_i}; \mathbf{R}^{sub}) = (h^{l,d}(\mathbf{q_i}; \mathbf{R}^{sub}) + h^{u,d}(\mathbf{q_i}; \mathbf{R}^{sub}))/2$$

$$= ((S^{-,d}(\mathbf{R}^{sub})k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) + S^{+,d}(\mathbf{R}^{sub})k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub}))) +$$
$$(S^{+,d}(\mathbf{R}^{sub})k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) + S^{-,d}(\mathbf{R}^{sub})k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub}))))/2 = S_{R,d}\bar{K}_h$$

and to the denominator $f(\mathbf{q_i}; \mathbf{R}^{sub})$ by: $f(\mathbf{q_i}; \mathbf{R}^{sub}) = |\mathbf{R}^{sub}|\bar{K}_h$. During recursion,

the bounds are tightened as:

$$\begin{aligned}
h^{l,d}(\mathbf{q_i}) += & S^{-,d}(\mathbf{R}^{sub})(k(d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})) - k(d^l(\mathbf{Q},\mathbf{R})))\\
& + S^{+,d}(\mathbf{R}^{sub})(k(d^u(\mathbf{Q}^{sub},\mathbf{R}^{sub})) - k(d^u(\mathbf{Q},\mathbf{R})))\\
h^{u,d}(\mathbf{q_i}) += & S^{+,d}(\mathbf{R}^{sub})(k(d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})) - k(d^l(\mathbf{Q},\mathbf{R})))\\
& + S^{-,d}(\mathbf{R}^{sub})(k(d^u(\mathbf{Q}^{sub},\mathbf{R}^{sub})) - k(d^u(\mathbf{Q},\mathbf{R})))\\
f^l(\mathbf{Q}^{sub}) += & |\mathbf{R}^{sub}|(k(d^u(\mathbf{Q}^{sub},\mathbf{R}^{sub})) - k(d^u(\mathbf{Q},\mathbf{R})))\\
f^u(\mathbf{q_i}) += & |\mathbf{R}^{sub}|(k(d^l(\mathbf{Q}^{sub},\mathbf{R}^{sub})) - k(d^l(\mathbf{Q},\mathbf{R})))
\end{aligned}$$

Specifying the CanSummarize Function. The global relative error bound τ is satisfied by ensuring a local pruning criterion in the function CanSummarize. Simple algebraic manipulation reveals that:

$$\begin{aligned}
&|\widetilde{\mathbf{h}}(\mathbf{q_i})/\widetilde{f}(\mathbf{q_i}) - \mathbf{h}(\mathbf{q_i})/f(\mathbf{q_i})|_1/|\mathbf{h}(\mathbf{q_i})/f(\mathbf{q_i})|_1 \leq \tau\\
\Leftrightarrow &|f(\mathbf{q_i})\widetilde{\mathbf{h}}(\mathbf{q_i}) - \widetilde{f}(\mathbf{q_i})\mathbf{h}(\mathbf{q_i})|_1 \leq \tau\widetilde{f}(\mathbf{q_i})|\mathbf{h}(\mathbf{q_i})|_1
\end{aligned}$$

Theorem 6.3.1 derives the pruning condition based on the triangle inequality, which shows how to satisfy the right hand side of the above relationship. The condition specifies the CanSummarize function for DT-MS to guarantee the global error bound. Multipole expansion is also used for more pruning [112]. We define some notations first. Given a query node $\mathbf{Q}^{sub}$, the bounds for $\mathbf{h}(\mathbf{q_i})$ in L_1 norm for any $\mathbf{q_i} \in \mathbf{Q}^{sub}$ are defined as: $L(\mathbf{Q}^{sub}) = \sum_{d=1}^{D} I(h^{l,d}(\mathbf{q_i}), h^{u,d}(\mathbf{q_i}))$, $U(\mathbf{Q}^{sub}) = \sum_{d=1}^{D} \max(|h^{l,d}(\mathbf{q_i})|, |h^{u,d}(\mathbf{q_i})|)$ where

$$I(a,b) = \begin{cases} a, a \geq 0 \\ -b, b < 0 \\ 0, otherwise \end{cases}$$

for $a, b \in \mathbb{R}$, such that $L(\mathbf{Q}^{sub}) \leq |\mathbf{h}(\mathbf{q_i})|_1 \leq U(\mathbf{Q}^{sub})$ for all $\mathbf{q_i} \in \mathbf{Q}^{sub}$.

Theorem 6.3.1. *Given a query node $\mathbf{Q}^{sub}$ and a reference node $\mathbf{R}^{sub}$, if $\mathbf{R}^{sub}$'s contribution to all $\mathbf{q_i} \in \mathbf{Q}^{sub}$ is approximated as $\widetilde{h}^d(\mathbf{q_i}; \mathbf{R}^{sub}) = (S_{R,d}(k(d^u(\mathbf{Q}^{sub},\mathbf{R}^{sub})) +$*

$k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub}))))/2, d = 1, ..., D$ *and* $\widetilde{f}_R(\mathbf{q_i}) = |\mathbf{R}^{sub}|\bar{K}_h$*, the following local pruning criterion must be enforced to guarantee the global relative error bound* τ*:* $k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub}))-k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) \leq \min\{\frac{\tau f^l(\mathbf{Q}^{sub})L(\mathbf{Q}^{sub})}{|\mathbf{R}|U(\mathbf{Q}^{sub})}, \frac{\tau L(\mathbf{Q}^{sub})}{\sum_d S^{A,d}}\}$

Proof. If the local pruning criterion is met and $\mathbf{R}^{sub}$ is approximated, we have

$$|\widetilde{h}^d(\mathbf{q_i}; \mathbf{R}^{sub}) - h^d(\mathbf{q_i}; \mathbf{R}^{sub})| \leq (h^{u,d}(\mathbf{q_i}; \mathbf{R}^{sub}) - h^{l,d}(\mathbf{q_i}; \mathbf{R}^{sub}))/2$$
$$= S^{A,d}(\mathbf{R}^{sub})(k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})))/2$$

for $d = 1, \cdots, D$ and $|\widetilde{f}_R(\mathbf{q_i}) - f(\mathbf{q_i})| \leq |\mathbf{R}^{sub}|(k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})))/2$. Given $\mathbf{q_i} \in \mathbf{Q}^{sub}$, suppose $\widetilde{h}(\mathbf{q_i})$ and $\widetilde{f}(\mathbf{q_i})$ were computed using reference nodes $\cup\{\mathbf{R}^{sub}\} = \mathbf{R}$. By the triangle inequality,

$$|f(\mathbf{q_i})\widetilde{h}(\mathbf{q_i}) - \widetilde{f}(\mathbf{q_i})\mathbf{h}(\mathbf{q_i})|_1 \leq |\widetilde{h}(\mathbf{q_i})|_1|f(\mathbf{q_i}) - \widetilde{f}(\mathbf{q_i})| + \widetilde{f}(\mathbf{q_i})|(\widetilde{h}(\mathbf{q_i}) - \mathbf{h}(\mathbf{q_i}))|_1$$
$$\leq \sum_{\mathbf{R}^{sub}} |\widetilde{h}(\mathbf{q_i})|_1(f(\mathbf{q_i}; \mathbf{R}^{sub}) - \widetilde{f}_R(\mathbf{q_i}))| + \widetilde{f}(\mathbf{q_i})|\widetilde{h}_R(\mathbf{q_i}) - \mathbf{h}(\mathbf{q_i}; \mathbf{R}^{sub})|_1$$
$$\leq |\widetilde{h}(\mathbf{q_i})|_1 \sum_{\mathbf{R}^{sub}} |\mathbf{R}^{sub}|(k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})))/2 +$$
$$\widetilde{f}(\mathbf{q_i}) \sum_{\mathbf{R}^{sub}} \sum_d S^{A,d}(\mathbf{R}^{sub})(k(d^l(\mathbf{Q}^{sub}, \mathbf{R}^{sub})) - k(d^u(\mathbf{Q}^{sub}, \mathbf{R}^{sub})))/2$$
$$\leq \sum_{\mathbf{R}^{sub}} \left[|\widetilde{h}(\mathbf{q_i})|_1 \frac{\tau|\mathbf{R}^{sub}|f^l(\mathbf{Q}^{sub})L(\mathbf{Q}^{sub})}{2NU(\mathbf{Q}^{sub})} + \widetilde{f}(\mathbf{q_i}) \frac{\tau \sum_d S^{A,d}(\mathbf{R}^{sub})L(\mathbf{Q}^{sub})}{2\sum_d S^{A,d}}\right] \leq \tau \widetilde{f}(\mathbf{q_i})|\mathbf{h}(\mathbf{q_i})|_1 \quad \square$$

$\square$

6.4 Experiments and Discussions

We have two tasks in the experiments. One is to compare the speedup of DT-MS over the naive MS. The other is to compare the speed, accuracy and stability in convergence among DT-MS, IFGT-MS and LSH-MS. We used the IFGT-MS and LSH-MS codes provided by the authors. LSH uses an Epanechnikov-like kernel. So we tested both the Gaussian kernel ($k(\mathbf{q_i} - \mathbf{r_j}) = e^{-\|\mathbf{q_i} - \mathbf{r_j}\|^2/2h^2}$) and Epanechnikov

Table 5: Running time (in seconds) of DT-MS and naive-MS with the Gaussian kernel. N_{it} is the number of iterations in MS, $\tau = 0.1$, $\epsilon = 0.01$.

Images	Speedup	Time(DT/Naive)	N_{it}	h_g
Fox	44.74	155.22/6944.54	1/1	0.0166
Snake	136.51	39.71/5420.36	1/1	0.0065
Cowboys	1.75	3059.38/5352.24	2/1	0.0172
Vase	19.06	300.66/5729.44	1/1	0.0163
Plane	32.86	187.54/6162.65	1/1	0.0102
Hawk	48.88	127.35/6224.48	1/1	0.0136

kernel ($k(\mathbf{q_i} - \mathbf{r_j}) = 1 - \|\mathbf{q_i} - \mathbf{r_j}\|^2/h^2$ if $\|\mathbf{q_i} - \mathbf{r_j}\| \leq h$, otherwise 0) for DT-MS. [3] $\mathbf{Q}$ is initialized as $\mathbf{R}$ for all the datasets.

Speedup of DT-MS over the Naive MS. We chose image segmentation as a representative clustering task. The goal of image segmentation is to cluster pixels into several distinct groups. We followed [201]'s approach of segmentation, where each datum represents the normalized CIE LUV color space for each pixel and the labels are assigned to the pixels by applying a k-means algorithm to the converged $\mathbf{Q}$ returned by MS. In other words, one image forms one dataset $\mathbf{R} \subset \mathbb{R}^3$ and the size of $\mathbf{R}$ equals the number of pixels in the image. We applied DT-MS and the naive MS to 10 test images from the Berkeley segmentation dataset.[4] The image size is 481×321, i.e. $N = 154,401$. The speedup is an order of magnitude in 7 images and two orders of magnitude in one image. A summary of running time and speedups for a set of representative images is given in Table 5. Segmentation results for these images are shown in Figure 38.

Comparison among DT-MS, IFGT-MS and LSH-MS. The speed, accuracy and stability in convergence of the three algorithms are empirically evaluated on synthetic and real datasets. The accuracy of convergence is evaluated by the relative

[3]The optimal bandwidth h_g for the Gaussian kernel is automatically selected by DT-KDE using leave-one-out least square cross validation. The optimal bandwidth h_e for the Epanechnikov kernel is determined as $h_e = 2.214 * h_g$ (for the univariate case) according to the equivalent kernel rescaling in Table 6.3 in [166].

[4]http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

error in L_1 norm as $|\widetilde{\mathbf{m}}(\mathbf{q_i}) - \mathbf{m}(\mathbf{q_i})|_1/|\mathbf{m}(\mathbf{q_i})|_1$, where $\mathbf{m}(\mathbf{q_i})$ is the final convergence of $\mathbf{q_i}$ using naive MS and $\widetilde{\mathbf{m}}(\mathbf{q_i})$ is produced by the approximation algorithm. Stable algorithms should exhibit low variance in the converged point positions over multiple runs. We demonstrate stability by displaying the results from different runs.

Experiment 1: We first compare the three methods on two typical images for segmentation (Figure 33). Table 6 shows the average running time and accuracy of convergence (represented in relative error) for two images. The results over different runs are not shown because the variations are mostly cancelled by applying k-means to group the converged pixels. LSH's running time has two parts: MS+(K, L) training. We include the training time because it is required for every dataset, and (K, L) training comprises the majority of the running time. DT-MS(Epan.) is the best in both speed and accuracy. The average number of iterations for IFGT-MS and DT-MS is very small (1 to 2) because the normalized CIE LUV space is sparse for the tested images.[5] IFGT-MS is faster than DT-MS(Gauss.) but has a slightly higher relative error. For segmentating images, such a difference can be ignored.

Table 6: Running time (in seconds) and relative error averaged over 3 runs. Top row: woman.ppm with $h_g = 0.027$, $h_e = 0.0598$. Bottom row: hand.ppm with $h_g = 0.0186$, $h_e = 0.0412$. $\epsilon = 0.01$, $\tau = 0.1$ for both images. IFGT-MS: $e = 4$, $p = 3$. DT-MS(Epan.) gives the best result in terms of speed and accuracy.

Alg.	Time	Rel. Err.
naive/DT(Epan.)	55.07/0.35	0/0
naive/DT(Gauss.)	194.6/2.08	0/0
IFGT	0.47	0.0093
LSH	$0.21 + 266.95$	0.3154
naive/DT(Epan.)	308.74/0.92	0/0
naive/DT(Gauss.)	1258.4/5.81	0/0
IFGT	1.24	$8.245e-5$
LSH	$0.52 + 621.15$	0.052501

[5]IFGT-MS often returns NaN because absolute error pruning creates zeroes in the numerator and the denominator of $\mathbf{m}(\mathbf{q_i})$.

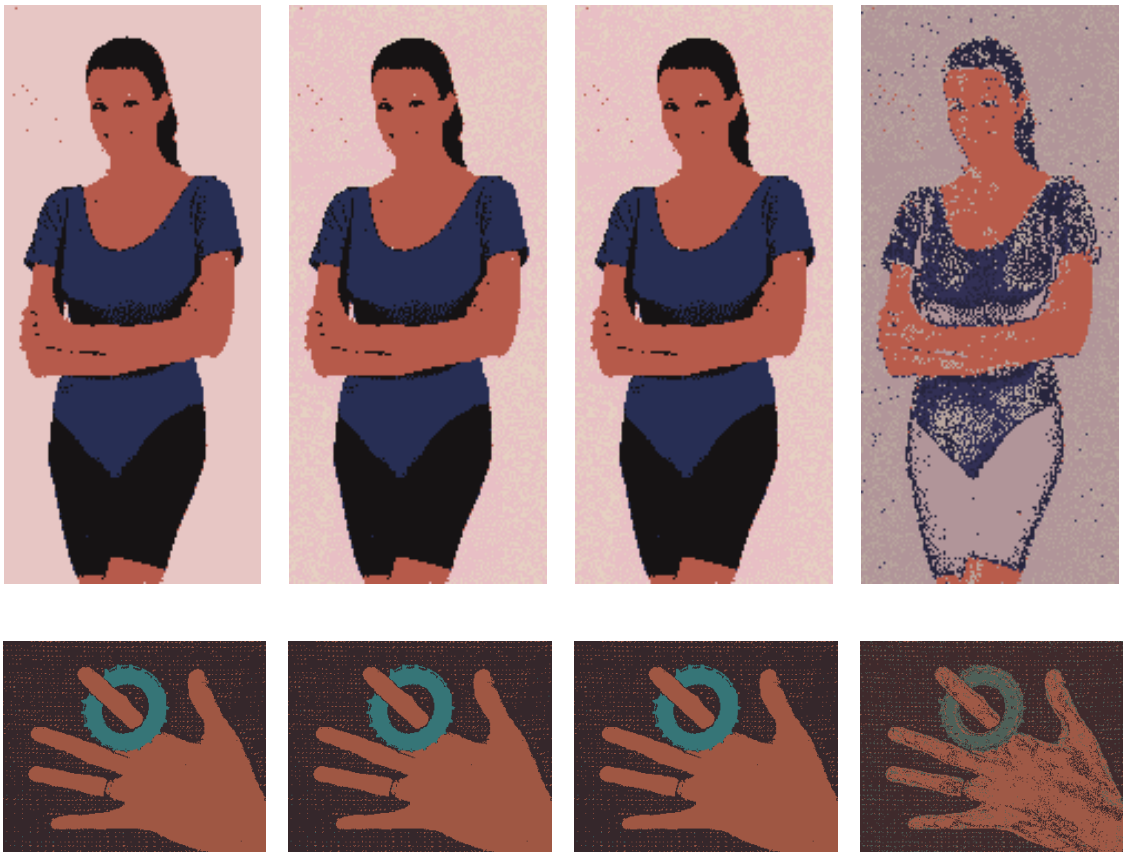


Figure 33: Size of woman: 116×261. Size of hand: 303×243.

Experiment 2: The segmentation is obtained by applying k-means to group the converged points. This is potentially a confounding factor, since k-means can compensate for poorly-converged points. Therefore we synthesized a dataset where k-means cannot work well, but MS can still find the correct modes. This experiment and the next one demonstrate MS's ability in noise reduction of the dataset to help reveal its intrinsic dimensionality [71]. Testing data containing 6000 2-D points was generated by adding Gaussian noise to sampled points on two intersected half circles (the blue dots in Figure 34), viewed as 2 c-shape clusters. Table 7 and Figure 34 again show that DT-MS(Epan.) achieves the best overall result among speed, accuracy and stability. IFGT-MS is slightly faster than DT-MS(Epan.) with slightly bigger variations in different runs. Naive-MS(Gauss.) runs faster than the DT-MS(Gauss.) for this dataset. This is because when the data points are not well clustered under certain

Table 7: Running time (in seconds) and relative error of convergence on 2-C-shape data averaged over 3 runs. $h_e = 8.856, h_g = 4$, $\epsilon = 0.2$, $\tau = 0.01$ for Epanechnikov kernel and $\tau = 0.001$ for Gaussian kernel. IFGT-MS: $e = 8$, $p = 30$. N_{it} is N/A for LSH-MS because it uses a different loop order from IFGT-MS and DT-MS.

Alg.	Time	N_{it}	Rel. Err.
naive/DT(Epan.)	38.56/11.89	22/22	0/1.8e-4
naive/DT(Gauss.)	190.21/207.6	26/26	0/1.16e-2
IFGT	10.74	25	0.015
LSH	0.58+279.73	N/A	0.1174

Table 8: Running time (in seconds) and relative error of convergence on noisyswiss-roll.ds averaged over 3 runs. $h_e = 4.06, h_g = 1.833$, $\epsilon = 0.02$, $\tau = 0.1$ for Epanechnikov kernel and $\tau = 0.01$ for Gaussian kernel. IFGT-MS: $e = 9$, $p = 20$. DT-MS(Epan.) is best in both speed and accuracy.

Alg.	Time	N_{it}	Rel Err.
naive/DT(Epan.)	992.39/148.16	44/44	0/1.5e-4
naive/DT(Gauss.)	4314.85/3116.9	51/51	0/0.025
IFGT	240.05	20	0.0573
LSH	3.81+713.58	N/A	0.2137

bandwidth, the pruning does not happen frequently enough to cancel the additional cost for distance computation per each query/reference node pair.

Experiment 3: Swissroll data with additive Gaussian noise ($\mathcal{N}(0, 4)$) (Figure 37).[6] $N = 20,000$, $D = 3$. Though both the number of points and the dimensionality are larger, DT-MS(Epan.) still achieves best performance in speed, accuracy and stability (Table 8 and Figure 35 and Figure 36).

Table 9: Running time (in seconds) and relative error of convergence on high-dimensional data averaged over 3 runs. $h_e = 0.49, h_g = 0.2212$, $\epsilon = 0.02$, $\tau = 0.1$ for both the Epanechnikov and Gaussian kernels. IFGT-MS: $e = 9$, $p = 7$. DT-MS(Epan.) gives the best result in terms of speed and accuracy.

Alg.	Time	N_{it}	Rel Err.
naive/DT(Epan.)	3515.74/516.34	7/7	0/0
naive/DT(Gauss.)	24189.8/39680	17/17	0/7.6e-6
IFGT	1260.56	2	0.2539
LSH	390.7+1026.9	N/A	0.4605

[6] http://isomap.stanford.edu/datasets.html

Experiment 4: High-dimensional data ($N = 68,040$, $D = 16$).[7] The running time and relative error of convergence are shown in Table 9. DT-MS(Epan.) again achieves the best performance in both speed and accuracy. We could improve IFGT-MS's relative error further by increasing p and e (which will increase the running time), but the algorithm failed due to memory limit. Even at its current level of accuracy, IFGT is slower than DT-MS(Epan.). DT-MS(Gauss.) is slower than the naive case for the same reason as explained in Experiment 2.

Summary of the Experiments. DT-MS with the Epanechnikov and the Gaussian kernels provides consistent and accurate convergence, and it is faster than naive MS (by two orders of magnitude in some cases with both kernels). DT-MS(Epan.) returns almost zero relative error when compared to the naive case. DT-MS(Gauss.) also returns zero relative error for well-clustered data (Table 6). For less well-clustered data, DT-MS(Gauss.) returns slightly bigger relative error than DT-MS(Epan.), but the error is small enough to be safely ignored (Table 8, 9).

DT-MS(Epan.) is always faster than DT-MS(Gauss.) in our datasets, because the Epanechnikov kernel has finite extent and can be pruned more frequently than the Gaussian kernel with zero approximation error [78]. The Epanechnikov kernel is also optimal in the sense of minimizing asymptotic mean integrated squared error, so it is statistically preferred. For some datasets the relative error for DT-MS(Gauss.) is bigger than τ (Table 7, 8). This is because τ controls the relative error of $\widetilde{m}(\mathbf{q_i})$ in one iteration of MS, not in the converged result. Thus, the approximated trajectory of a point may not match the one computed by the naive method.

DT-MS(Epan.) always dominates IFGT-MS and LSH-MS in speed, accuracy and stability. In addition, DT-MS(Epan.) requires no parameter tuning. IFGT-MS can achieve very good speedup and accuracy, if the parameters are set correctly (Table 7 and Figure 34). LSH-MS with an adequate (K, L) pair is very fast. However,

[7] http://www.ics.uci.edu/∼kdd/databases/CorelFeatures/CorelFeatures.data.html

training (K, L) takes much time and is dependent on the dataset and the search range of (K, L). Both IFGT-MS and LSH-MS require trial-and-error, manual tuning of parameters and also require much more storage than DT-MS.

6.5 Conclusion

This chapter presents a new algorithm DT-MS for accelerating mean shift. It extends the dual-tree method to the fast approximation of the signed mean vector in MS. Our experiments have demonstrated its fast, accurate and stable approximation of MS. Especially with the Epanechnikov kernel, DT-MS scales quite well to larger datasets with higher dimensions. It has the best performance in terms of speed, accuracy and stability in comparison to IFGT-MS, LSH-MS and DT-MS(Gauss.). We note that there is an interesting follow-up work [187].

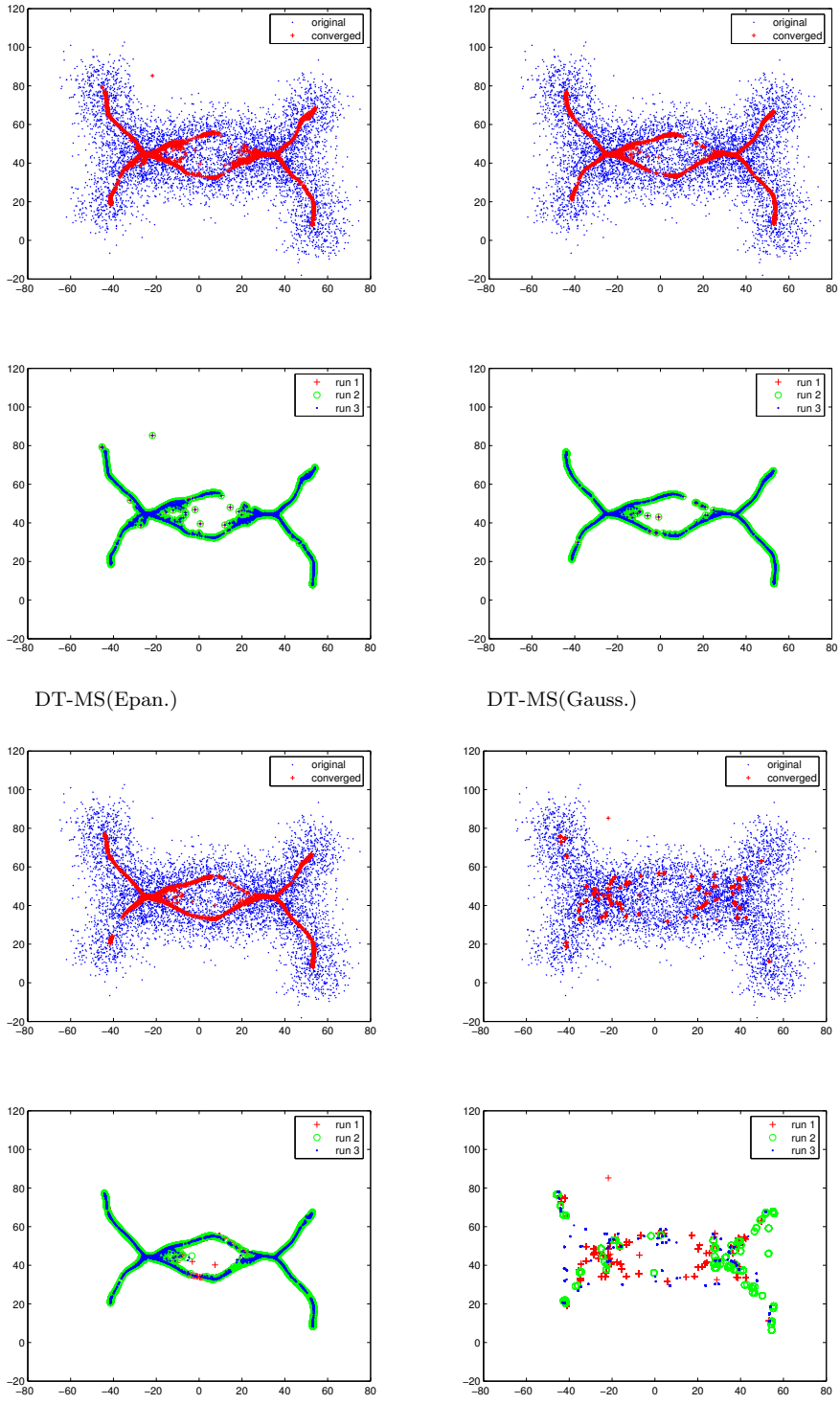


Figure 34: Accuracy/stability of convergence. Converged queries (red) imposed on the original data (blue). Stability illustrated by the converged queries of 3 runs(indicated by 3 colors).

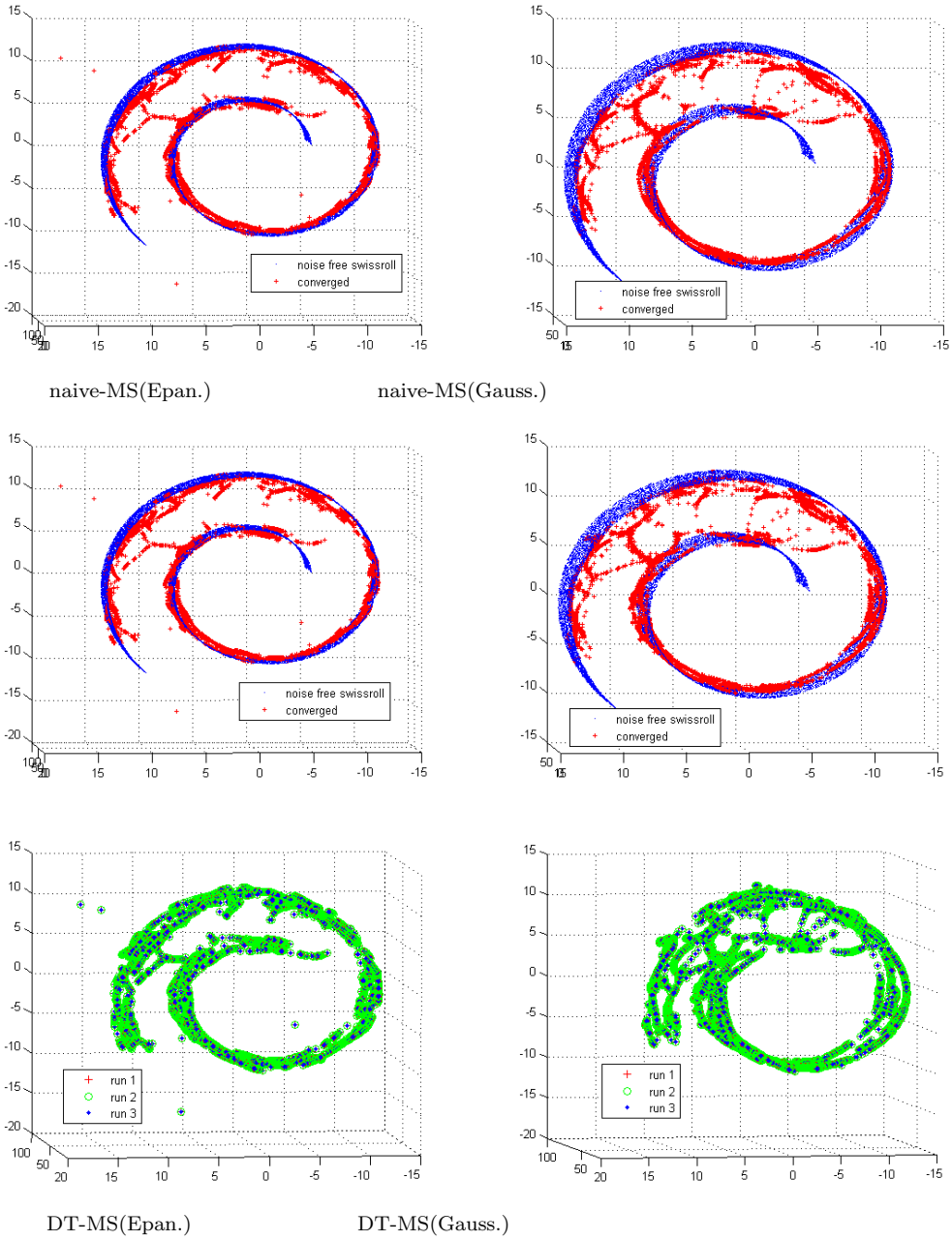


Figure 35: Accuracy and stability of convergence: For clarity all the MS results (red) are imposed on the original swissroll (blue). Stability is illustrated by the converged queries of 3 runs (indicated by 3 colors). For comparison, the results obtained by the naive MS are shown in the top row.

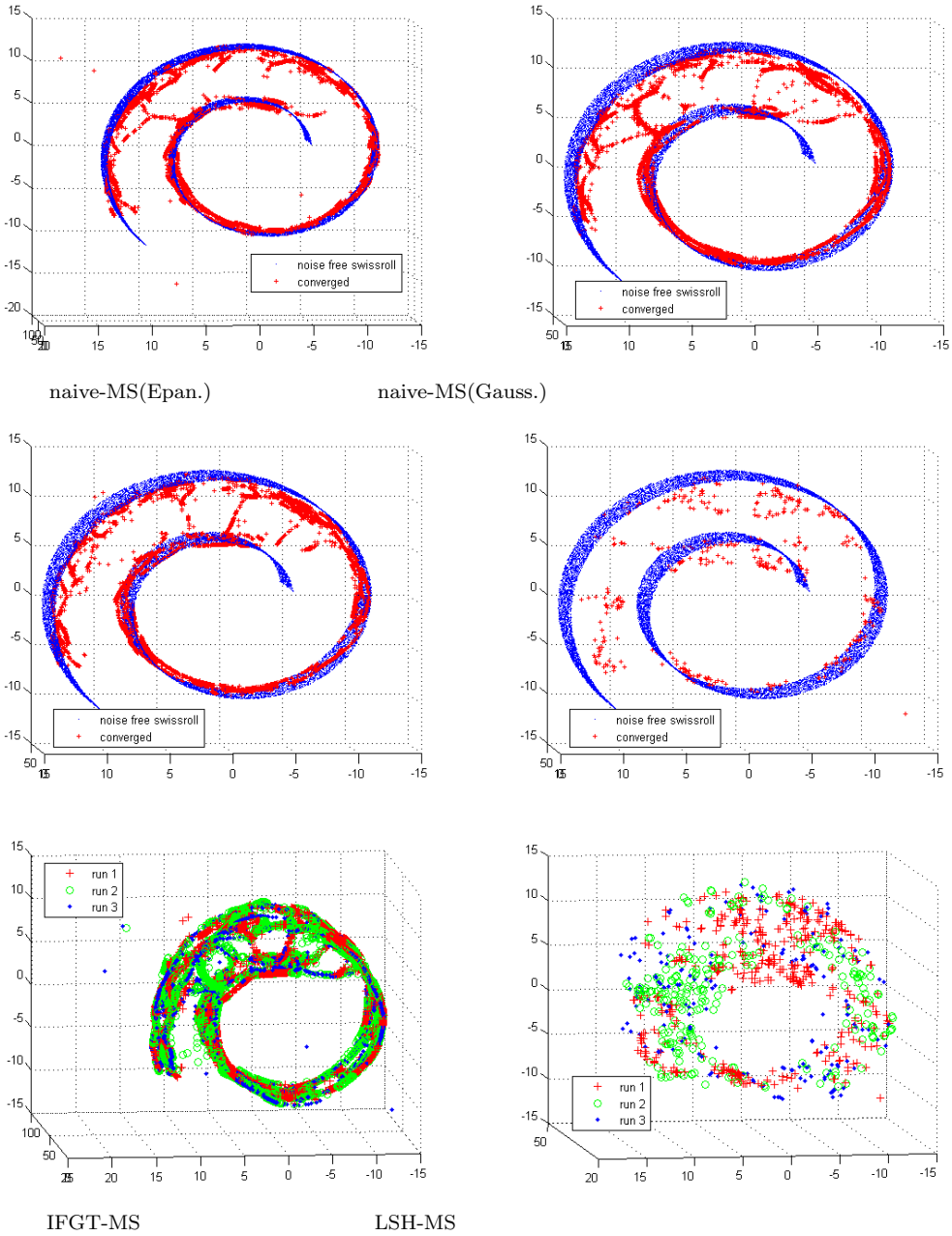


Figure 36: Accuracy and stability of convergence: For clarity all the MS results (red) are imposed on the original swissroll (blue). Stability is illustrated by the converged queries of 3 runs (indicated by 3 colors). For comparison, the results obtained by the naive MS are shown in the top row.

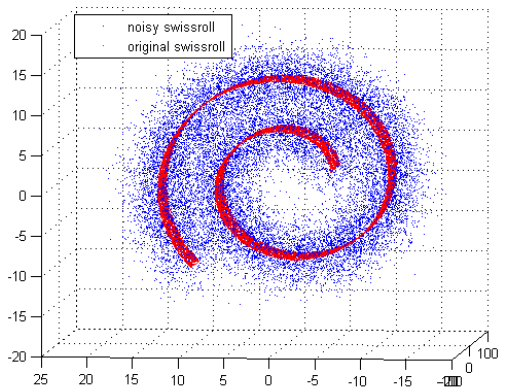


Figure 37: Noisy swissroll (in blue) and the clean swissroll (in red).

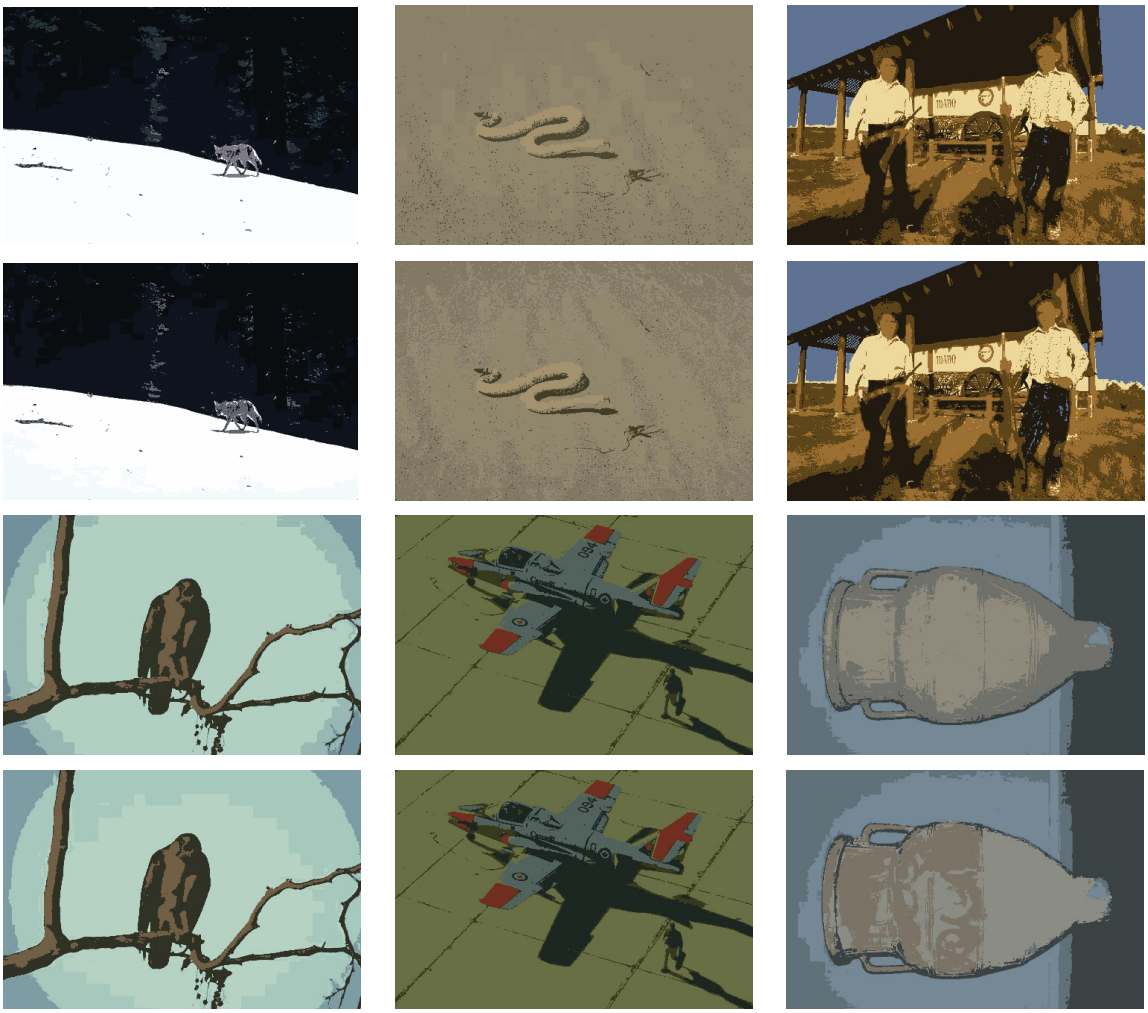


Figure 38: For each image segmentation pair, top: DT-MS, bottom: naive-MS.

CHAPTER VII

BEYOND PAIRWISE INTERACTIONS: FAST SUMMATION METHODS FOR N-TUPLE CONTINUOUS FUNCTIONS

In this chapter, we generalize previous algorithmic frameworks for rapidly computing pair-wise summations to include higher-order summations. Suppose we are given a set of particles $X = \{\mathbf{x_1}, \cdots, \mathbf{x_N}\}$ in D-dimensional space. We first define the computational problem to be tackled.

Problem: For $\mathbf{x} \in \mathbf{X}$ and a n-tuple function $\phi : \underbrace{\mathbb{R}^D \times \cdots \times \mathbb{R}^D}_{n \text{ copies}} \to \mathbb{R}$, the probability guarantee $0 < \alpha \leq 1$, the relative error level $\epsilon > 0$, and the following form[1]:

$$\Phi(\mathbf{x}; \underbrace{\mathbf{X} \times \cdots \times \mathbf{X}}_{(n-1) \text{ copies}}) = \sum_{\mathbf{x_{i_2}} \in \mathbf{X} \setminus \{\mathbf{x}\}} \sum_{\substack{\mathbf{x_{i_3}} \in \mathbf{X} \setminus \{\mathbf{x}\} \\ i_2 < i_3}} \cdots \sum_{\substack{\mathbf{x_{i_n}} \in \mathbf{X} \setminus \{\mathbf{x}\} \\ i_{n-1} < i_n}} \phi(\mathbf{x}, \mathbf{x_{i_2}}, \cdots, \mathbf{x_{i_n}}) \tag{7.0.1}$$

Task: Compute an approximation $\widetilde{\Phi}(\mathbf{x}; \underbrace{\mathbf{X} \times \cdots \times \mathbf{X}}_{(n-1) \text{ copies}})$ for each $\mathbf{x} \in \mathbf{X}$ such that $|\widetilde{\Phi}(\mathbf{x}; \underbrace{\mathbf{X} \times \cdots \times \mathbf{X}}_{(n-1) \text{ copies}}) - \Phi(\mathbf{x}; \underbrace{\mathbf{X} \times \cdots \times \mathbf{X}}_{(n-1) \text{ copies}})| \leq \epsilon \Phi(\mathbf{x}; \underbrace{\mathbf{X} \times \cdots \times \mathbf{X}}_{(n-1) \text{ copies}})$ as fast as possible.

Sums of the form Equation (7.0.1) occur in molecular dynamics, protein structure prediction, and other similar contexts. Biomolecular simulations usually break down the interactions in complex chemical systems into balls-and-springs mechanical models augmented by torsional terms, pairwise point charge electrostatic terms, and simple pairwise dispersion (van der Waals) interactions, etc. However, such pairwise ($n = 2$)

[1] In computing $\Phi(\mathbf{x})$, we fix one of the arguments of ϕ as $\mathbf{x}$ and choose a $(n-1)$-subset from $\mathbf{X}^{(n-1)}$ which does not contain $\mathbf{x}$.

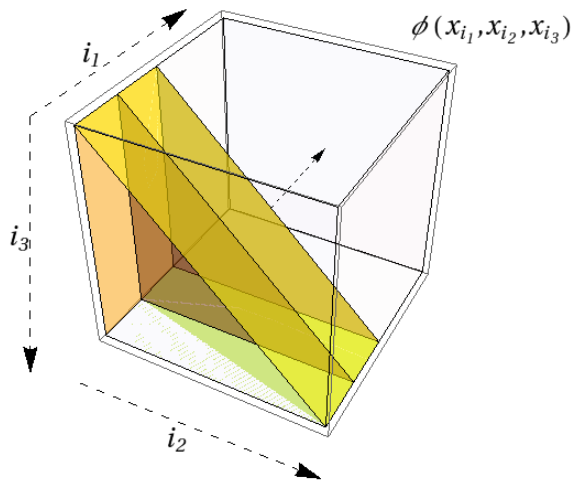


Figure 39: An example multibody computation ($n = 3$). For each fixed argument $\mathbf{x_{i_1}}$, $\Phi(\mathbf{x_{i_1}})$ equals the summation of the entries $\phi(\mathbf{x_{i_1}}, \mathbf{x_{i_2}}, \mathbf{x_{i_3}})$ in the shaded region corresponding to $\mathbf{x_{i_1}}$.

interactions often fail to capture important, complex non-additive interactions found in real systems. Though many researchers have argued that multibody potentials enable more accurate and realistic molecular modeling, the evaluation of n-body forces for $n \geq 3$ in systems beyond tiny sizes (less than 10,000 particles) has not been possible due to the unavailability of an efficient way to realize the computation.

In this chapter we focus on computing multibody potentials of the third order ($n = 3$), but frame our presentation so that the methods can easily be generalized to handle higher-order potentials. For concreteness, we consider the Axilrod-Teller potential (dispersion potential):

$$\phi(\mathbf{x_i}, \mathbf{x_j}, \mathbf{x_k}) = \frac{1 + 3\cos\theta_i \cos\theta_j \cos\theta_k}{||\mathbf{x_i} - \mathbf{x_j}||^3 ||\mathbf{x_i} - \mathbf{x_k}||^3 ||\mathbf{x_j} - \mathbf{x_k}||^3} \tag{7.0.2}$$

where θ_i, θ_j, θ_k are the angles at the vertices of the triangle $\mathbf{x_i x_j x_k}$ and $||\cdot||$ is the Euclidean distance metric. This potential [9] describes induced dipole interactions between triples of atoms, and is known to be important for the accurate computation of the physical properties of certain noble gases.

For the first time, we introduce a fast algorithm for efficiently computing multibody potentials for a large number of particles. We restrict the class of multibody

potentials to those that can be factorized as products of functions of pairwise Euclidean distances. That is,

$$\phi(\mathbf{x_{i_1}}, \cdots, \mathbf{x_{i_n}}) = \prod_{1 \le p < q \le n} \phi_{p,q}(\mathbf{x_{i_p}}, \mathbf{x_{i_q}}) = \prod_{1 \le p < q \le n} \phi_{p,q}(||\mathbf{x_{i_p}} - \mathbf{x_{i_q}}||) \tag{7.0.3}$$

We extend the analytic series-expansion-based approach in [37, 86, 84] to handle potential functions that describe n-body interactions with $n > 2$. Our algorithm can compute multibody potentials within a user-specified error level.

Our work utilizes and extends a framework for efficient algorithms for so-called *generalized N-Body Problems* [78], which introduced *multi-tree methods*. The framework was originally developed to accelerate common bottleneck statistical computations based on distances, utilizing multiple *kd*-trees and other spatial data structures to reduce computation times both asymptotically and practically by multiple orders of magnitude. This work extends the framework with higher-order hierarchical series approximation techniques, demonstrating a fast multipole-type method for higher-order interactions for the first time, effectively creating a *Multibody Multipole Method*.

Section 7.2 introduces the *generalized N-body framework* and describes a partial extension of fast multipole-type methods to handle higher-order interactions; we will discuss the technical difficulties for deriving all of the necessary tools for the general multibody case. As a result, we utilize only a simple but effective approximation using the center-of-mass approximations. Section 7.3 focuses on three-body interactions, introduces methods to do potential computations under both deterministic and probabilistic error criteria, and provides a description of the fast algorithm for the three-body case. Section 7.4 proves that our proposed algorithms can approximate potentials within user-specified error bounds. Section 7.5 shows experimental scalability results for our proposed algorithms against the naive algorithm, under different error parameter settings.

7.1 Related Work

A series of papers first laid the foundations for efficiently computing sums of pairwise potentials such as Coulombic and Yukawa potentials [37, 86, 84]. The common approach in these papers is to derive analytical series expansions of the given potential function in either Cartesian or spherical coordinate systems. The series expansion is then truncated after taking a fixed number of terms. The associated error bounds are derived from summing the truncated terms in an appropriate infinite geometric sum or bounding the remainder term using Taylor's theorem. A recent line of work on efficient computation of pairwise function has focused on developing numerical representations of the potential matrix $[\phi(\mathbf{x_m}, \mathbf{x_n})]_{m,n=1}^{N}$, rather than relying on analytical expansion of the potential function. [129] and [99] use singular value decomposition and the QR decomposition to compute the compressed forms of the potential function and the three translation operators. [5, 203] take the "pseudo-particle" approach by placing equivalent artificial charges on the bounding surface of the actual particles by solving appropriate integral equations. All of these works have been limited to pairwise potential functions, and the approach does not naturally suggest a generalization to n-body potentials with $n > 2$. To our knowledge, no research has been performed on the problem of evaluating multibody potentials using a method more sophisticated than the $\mathcal{O}(N^n)$ brute-force algorithm with an ad-hoc cut-off distance. [127, 126].

7.2 Generalized N-body Framework

We use a variant of *kd*-trees [15] to form hierarchical groupings of points based on their locations using the recursive procedure shown in Algorithm 2.2.1. We note that the cost of building a *kd*-tree is negligible compared to the actual multibody potential computation (see Section 7.5). See Figure 7.

Step 2 in the algorithm listed in the preliminary chapter utilizes the procedure shown in Algorithm 7.2.1 (called by setting each $\mathbf{P_i} = \mathbf{X}$ for $1 \leq i \leq n$), a recursive

Algorithm 7.2.1 MTPotentialCanonical($\{\mathbf{P_i}\}_{i=1}^n$)

if CanSummarize($\{\mathbf{P_i}\}_{i=1}^n$) (Try approximation.) **then**
  Summarize($\{\mathbf{P_i}\}_{i=1}^n, \epsilon, \tau, \alpha$)
else
  if all of S_i are leaves **then**
    MTPotentialBase($\{\mathbf{P_i}\}_{i=1}^n$) (Base case.)
  else
    Find an internal node $\mathbf{P_k}$ to split among $\{\mathbf{P_i}\}_{i=1}^n$.
    Propagate bounds of $\mathbf{P_k}$ to ${\mathbf{P_k}}^L$ and ${\mathbf{P_k}}^R$.
    MTPotentialCanonical($\{\mathbf{P_1}, \cdots, \mathbf{P_{k-1}}, {\mathbf{P_k}}^L, \mathbf{P_{k+1}}, \cdots, \mathbf{P_n}\}$)
    MTPotentialCanonical($\{\mathbf{P_1}, \cdots, \mathbf{P_{k-1}}, {\mathbf{P_k}}^R, \mathbf{P_{k+1}}, \cdots, \mathbf{P_n}\}$)
    Refine summary statistics based on the two recursive calls.

function that allows us to consider the n-tuples formed by choosing each $\mathbf{x_i}$ from $\mathbf{P_i}$; we can gain efficiency over the naive enumeration of the n-tuples by using the bounding box and the moment information stored in each $\mathbf{P_i}$. One such information is the distance bound computed using the bounding box (see Figure 10). CanSummarize function first eliminates redundant recursive calls for the list of node tuples that satisfy the following condition: if there exists a pair of nodes $\mathbf{P_i}$ and $\mathbf{P_j}$ ($i < j$) among the node list $\mathbf{P_1}, \cdots, \mathbf{P_n}$, such that the maximum depth-first rank of $\mathbf{P_i}$ is less than the minimum depth-first rank of $\mathbf{P_j}$. In this case, the function returns true. See Figure 7 and [132]. In addition, if any one of the nodes in the list includes one of the other nodes (i.e. there exists nodes $\mathbf{P_i}$ and $\mathbf{P_j}$ such that the minimum depth-first rank of $\mathbf{P_i}$ < the minimum depth-first rank of $\mathbf{P_j}$ < the maximum depth-first rank of $\mathbf{P_j}$ < the maximum depth-first rank of $\mathbf{P_i}$), CanSummarize returns false. We do this because it is a bit tricky to count the number of tuples for each point in this case (see Figure 40).

Otherwise, CanSummarize function tests whether each potential sum for $\mathbf{x} \in$

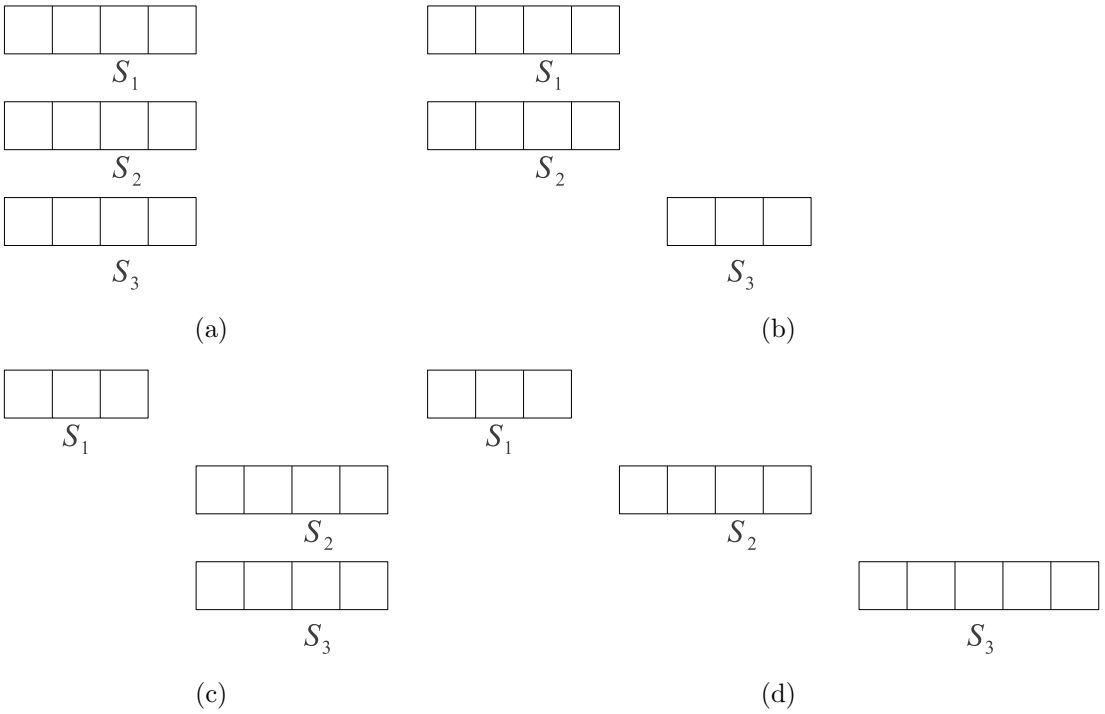


Figure 40: For $n = 3$, four canonical cases of the three "valid" (i.e. the particle indices in each node are in increasing depth-first order) node tuples encountered during the algorithm: (a) All three nodes are equal; (b) $\mathbf{S_1}$ and $\mathbf{S_2}$ are equal, and $\mathbf{S_3}$ comes later in the depth-first order; (c) $\mathbf{S_2}$ and $\mathbf{S_3}$ are equal and come later in the depth-first order; (d) All three nodes are different.

$\bigcup_{1 \le m \le n} \mathbf{P_m}$ can be approximated within the error tolerance determined by the algorithm. For example, if $n = 4$, we test for each $\mathbf{x_1} \in \mathbf{P_1}$, $\mathbf{x_2} \in \mathbf{P_2}$, $\mathbf{x_3} \in \mathbf{P_3}$, $\mathbf{x_4} \in \mathbf{P_4}$,

$$\Phi(\mathbf{x_1}; \mathbf{P_2} \times \mathbf{P_3} \times \mathbf{P_4}) = \sum_{\mathbf{x_{i_2}} \in \mathbf{P_2} \setminus \{\mathbf{x_1}\}} \sum_{\substack{\mathbf{x_{i_3}} \in \mathbf{P_3} \setminus \{\mathbf{x_1}\} \\ i_2 < i_3}} \sum_{\substack{\mathbf{x_{i_4}} \in \mathbf{P_4} \setminus \{\mathbf{x_1}\} \\ i_3 < i_4}} \phi(\mathbf{x_1}, \mathbf{x_{i_2}}, \mathbf{x_{i_3}}, \mathbf{x_{i_4}})$$

$$\Phi(\mathbf{x_2}; \mathbf{P_1} \times \mathbf{P_3} \times \mathbf{P_4}) = \sum_{\mathbf{x_{i_1}} \in \mathbf{P_1} \setminus \{\mathbf{x_2}\}} \sum_{\substack{\mathbf{x_{i_3}} \in \mathbf{P_3} \setminus \{\mathbf{x_2}\} \\ i_1 < i_3}} \sum_{\substack{\mathbf{x_{i_4}} \in \mathbf{P_4} \setminus \{\mathbf{x_2}\} \\ i_3 < i_4}} \phi(\mathbf{x_2}, \mathbf{x_{i_1}}, \mathbf{x_{i_3}}, \mathbf{x_{i_4}})$$

$$\Phi(\mathbf{x_3}; \mathbf{P_1} \times \mathbf{P_2} \times \mathbf{P_4}) = \sum_{\mathbf{x_{i_1}} \in \mathbf{P_1} \setminus \{\mathbf{x_3}\}} \sum_{\substack{\mathbf{x_{i_2}} \in \mathbf{P_2} \setminus \{\mathbf{x_3}\} \\ i_1 < i_2}} \sum_{\substack{\mathbf{x_{i_4}} \in \mathbf{P_4} \setminus \{\mathbf{x_3}\} \\ i_2 < i_4}} \phi(\mathbf{x_3}, \mathbf{x_{i_1}}, \mathbf{x_{i_2}}, \mathbf{x_{i_4}})$$

$$\Phi(\mathbf{x_4}; \mathbf{P_1} \times \mathbf{P_2} \times \mathbf{P_3}) = \sum_{\mathbf{x_{i_1}} \in \mathbf{P_1} \setminus \{\mathbf{x_4}\}} \sum_{\substack{\mathbf{x_{i_2}} \in \mathbf{P_2} \setminus \{\mathbf{x_4}\} \\ i_1 < i_2}} \sum_{\substack{\mathbf{x_{i_3}} \in \mathbf{P_3} \setminus \{\mathbf{x_4}\} \\ i_2 < i_3}} \phi(\mathbf{x_4}, \mathbf{x_{i_1}}, \mathbf{x_{i_2}}, \mathbf{x_{i_3}})$$

can be approximated. If the approximation is not possible, then the algorithm continues to consider the data at a finer granularity; it chooses an internal node $\mathbf{P_k}$

(typically the one with the largest diameter) to split among $\{\mathbf{P_i}\}_{i=1}^{n}$. Before recursing to two sub-calls in Line 9 and Line 10 of Algorithm 7.2.1, the algorithm can optionally push quantities from a node that is being split to its child nodes (Line 8). After returning from the recursive calls, the node that was just split can refine *summary statistics* based on the results accumulated on its child nodes. The details of these operations are available in earlier papers [78, 82, 80, 77, 115].

The basic idea is to terminate the recursion as soon as possible, i.e. by considering a tuple of large subsets and avoiding the number of exhaustive leaf-leaf-leaf computations. We note that the CanSummarize and Summarize functions effectively replace unwieldy interaction lists used in FMM algorithms. Interaction lists in n-tuple interaction, if naively enumerated, can be large depending on the potential function ϕ and the dimensionality D of the problem, whereas the generalized N-body approach can handle a wide spectrum of problems without this drawback.

7.2.1 Algorithm for Pairwise Potentials ($n = 2$)

The general algorithmic strategy for pairwise potentials $\phi(\cdot, \cdot)$ is described in [78, 82, 80, 77], and consists of the following three main phases. Suppose we are given a set of "source" points (denoted as *reference points*) and a set of "target" points (denoted as *query points*).

1. **Bottom-up phase:** Compute far-field moments of order p in every leaf node of the reference tree. The resulting far-field expansion of each reference node $\mathbf{P_2}$ is given by:

$$\begin{aligned}\Phi(\mathbf{x}; \mathbf{P_2}) &= \sum_{\boldsymbol{\alpha} \geq 0} \left[\sum_{\mathbf{x_{i_2}} \in \mathbf{P_2}} \frac{(-1)^{\boldsymbol{\alpha}}}{\boldsymbol{\alpha}!} (\mathbf{x_{i_2}} - \mathbf{c_{P_2}})^{\boldsymbol{\alpha}} \right] D^{\boldsymbol{\alpha}} \phi(\mathbf{x} - \mathbf{c_{P_2}}) \\ &= \sum_{\boldsymbol{\alpha} \geq 0} M_{\boldsymbol{\alpha}}(\mathbf{P_2}, \mathbf{c_{P_2}}) D^{\boldsymbol{\alpha}} \phi(\mathbf{x} - \mathbf{c_{P_2}}) \end{aligned} \tag{7.2.1}$$

   $\Phi(\mathbf{x}; \mathbf{P_2})$ reads as "the potential sum on x due to the contribution of $\mathbf{P_2}$" and $M_{\boldsymbol{\alpha}}(\mathbf{P_2}, \mathbf{c_{P_2}})$ as "the $\boldsymbol{\alpha}$-th far-field coefficient of $\mathbf{P_2}$ centered at $\mathbf{c_{P_2}}$." Because it

is impossible to store an infinite number of far-field moments $M_{\boldsymbol{\alpha}}(\mathbf{P_2}, \mathbf{c_{P_2}})$, we truncate the Taylor expansion up to the order p (determined either arbitrarily or by an appropriate error criterion):

$$\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}; \mathsf{F}(\mathbf{c_{P_2}}, p)) = \sum_{|\boldsymbol{\alpha}| \leq p} M_{\boldsymbol{\alpha}}(\mathbf{P_2}, \mathbf{c_{P_2}}) D^{\boldsymbol{\alpha}} \phi(\mathbf{x} - \mathbf{c_{P_2}}) \tag{7.2.2}$$

such that $\left| \widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}) - \Phi(\mathbf{x}; \mathbf{P_2}) \right|$ is sufficiently small. $\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}; \mathsf{F}(\mathbf{c_{P_2}}, p))$ reads as "the approximated potential sum on x due to the points owned by $\mathbf{P_2}$ using up to the p-th order far-field expansion of $\mathbf{P_2}$ centered at $\mathbf{c_{P_2}}$."

For internal reference nodes, perform the far-to-far (F2F) translation to convert the far-field moments owned by the child nodes to form the far-field moments for their common parent node $\mathbf{P_2}$. For example, the far-field moments of P_2^L centered at $\mathbf{c}_{P_2^L}$ is shifted to $\mathbf{c_{P_2}}$ by:

$$\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}^L; \mathsf{F}(\mathbf{c_{P_2}}, p)) = \sum_{\boldsymbol{\gamma} \leq p} M_{\boldsymbol{\gamma}}(\mathbf{P_2}^L, \mathbf{c_{P_2}}) (-1)^{\boldsymbol{\gamma}} D^{\boldsymbol{\gamma}} \phi(\mathbf{x} - \mathbf{c_{P_2}}) \tag{7.2.3}$$

where

$$M_{\boldsymbol{\gamma}}(\mathbf{P_2}^L, \mathbf{c_{P_2}}) = \sum_{\boldsymbol{\alpha} \leq \boldsymbol{\gamma}} \frac{M_{\boldsymbol{\alpha}}(\mathbf{P_2}^L, \mathbf{c}_{\mathbf{P_2}^L})(\mathbf{c}_{\mathbf{P_2}^L} - \mathbf{c_{P_2}})^{\boldsymbol{\gamma}-\boldsymbol{\alpha}}}{(\boldsymbol{\gamma} - \boldsymbol{\alpha})!} \tag{7.2.4}$$

Note that there is no error incurred in each F2F translation, i.e. $\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}^L; \mathsf{F}(\mathbf{c}_{\mathbf{P_2}^L}, p)) = \widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}^L; \mathsf{F}(\mathbf{c_{P_2}}, p))$ for any query point y from the intersection of the domains of x for $\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}^L; \mathsf{F}(\mathbf{c}_{\mathbf{P_2}^L}, p))$ and $\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}^L; \mathsf{F}(\mathbf{c_{P_2}}, p))$; the domain for which the far-field expansion remains valid depends on the error bound criterion for each potential. The far-field moments of the parent node $\mathbf{P_2}$ is the sum of the translated moments of its child nodes: $M_{\boldsymbol{\gamma}}(\mathbf{P_2}, \mathbf{c_{P_2}}) = \sum_{\boldsymbol{\alpha} \leq \boldsymbol{\gamma}} \frac{M_{\boldsymbol{\alpha}}(\mathbf{P_2}^L, \mathbf{c}_{\mathbf{P_2}^L})(\mathbf{c}_{\mathbf{P_2}^L} - \mathbf{c_{P_2}})^{\boldsymbol{\gamma}-\boldsymbol{\alpha}}}{(\boldsymbol{\gamma}-\boldsymbol{\alpha})!} + \frac{M_{\boldsymbol{\alpha}}(\mathbf{P_2}^R, \mathbf{c}_{\mathbf{P_2}^R})(\mathbf{c}_{\mathbf{P_2}^R} - \mathbf{c_{P_2}})^{\boldsymbol{\gamma}-\boldsymbol{\alpha}}}{(\boldsymbol{\gamma}-\boldsymbol{\alpha})!}$

2. **Approximation phase:** For a given pair of the query and the reference nodes, determine the order of approximation and either (1) translate the far-field moments of the reference node to the local moments of the query node (2) or

recurse to their subsets, if the F2L translation is more costly than the direct exhaustive method.

Let us re-write the exact contribution of $\mathbf{P_2}$ to a point $\mathbf{x} \in \mathbf{P_1}$:

$$\begin{aligned}\Phi(\mathbf{x}; \mathbf{P_2}) &= \sum_{\boldsymbol{\beta} \geq 0} \frac{1}{\boldsymbol{\beta}!} \sum_{\boldsymbol{\alpha} \geq 0} M_{\boldsymbol{\alpha}}(\mathbf{P_2}, \mathbf{c_{P_2}}) D^{\boldsymbol{\alpha}+\boldsymbol{\beta}} \phi(\mathbf{c_{P_1}} - \mathbf{c_{P_2}})(\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\beta}} \\ = \sum_{\boldsymbol{\beta} \geq 0} \left[\sum_{\mathbf{x_{i_2}} \in \mathbf{P_2}} \frac{1}{\boldsymbol{\beta}!} D^{\boldsymbol{\beta}} \phi(\mathbf{c_{P_1}} - \mathbf{x_{i_2}}) \right] (\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\beta}} &= \sum_{\boldsymbol{\beta} \geq 0} N_{\boldsymbol{\beta}}(\mathbf{P_2}, \mathbf{c_{P_1}})(\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\beta}}\end{aligned} \tag{7.2.5}$$

where $N_{\boldsymbol{\beta}}(\mathbf{P_2}, \mathbf{c_{P_1}})$ reads as "the exact local moments [2] contributed by the points in $\mathbf{P_2}$ centered at $\mathbf{c_{P_1}}$." Truncating Equation (7.2.5) at $|\boldsymbol{\beta}| \leq p'$ for some $p' \leq p$ yields a direct local accumulation of order p.

From the bottom-up phase, we know that $|\boldsymbol{\alpha}| \leq p$. Similarly, we can store only a finite number of local moments up to the order $p' \leq p$ and thus $|\boldsymbol{\beta}| \leq p'$. We get the local expansion for $\mathbf{P_1}$ formed due to translated far-field moments of $\mathbf{P_2}$:

$$\begin{aligned}\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2}; \widetilde{N}(\mathbf{c_{P_1}}, p')) &= \sum_{|\boldsymbol{\beta}| \leq p'} \left[\frac{1}{\boldsymbol{\beta}!} \sum_{|\boldsymbol{\alpha}| \leq p'} M_{\boldsymbol{\alpha}}(\mathbf{P_2}, \mathbf{c_{P_2}}) D^{\boldsymbol{\alpha}+\boldsymbol{\beta}} \phi_{1,2}(\mathbf{c_{P_1}} - \mathbf{c_{P_2}}) \right] (\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\beta}} \\ &= \sum_{|\boldsymbol{\beta}| \leq p'} \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{P_2}, \mathbf{c_{P_1}})(\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\beta}}\end{aligned} \tag{7.2.6}$$

where $\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{P_2}, \mathbf{c_{P_1}})$ reads as "approximation to the exact local moments $N_{\boldsymbol{\beta}}(\mathbf{P_2}, \mathbf{c_{P_1}})$" and $\widetilde{\Phi}(x; \mathbf{P_2}; \widetilde{N}(\mathbf{c_{P_1}}, p'))$ as "the approximated potential sum on x due to the points in $\mathbf{P_2}$ using up to the p-th order inexact local moments centered at $\mathbf{c_{P_1}}$". The F2L translation is applied only if $\left| \widetilde{\Phi}(x; \mathbf{P_2}; \widetilde{N}(\mathbf{c_{P_1}}, p')) - \Phi(\mathbf{x}; \mathbf{P_2}) \right|$ is sufficiently small.

3. **Top-down phase:** Propagate the local moments of each query node (i.e. pruned quantities) to its child nodes using the local-to-local (L2L) operator.

[2] We use N to denote the local moments because a "near-field" expansion is another widely used term for a local expansion. It avoids the potential notational confusion in the later parts of the paper.

Suppose we have the following local expansion for $\mathbf{x} \in \mathbf{P_1}$:

$$\widetilde{\Phi}(\mathbf{x}; F2L(\mathbf{P_1}) \cup DL(\mathbf{P_1}); \widetilde{N}(\mathbf{c_{P_1}}, p^u_{\mathbf{P_1}})) = \sum_{|\boldsymbol{\alpha}| \leq p^u_{\mathbf{P_1}}} \widetilde{N}_{\boldsymbol{\alpha}}(F2L(\mathbf{P_1}) \cup DL(\mathbf{P_1}), \mathbf{c_{P_1}})(\mathbf{x_{i_1}} - \mathbf{c_{P_1}})^{\boldsymbol{\alpha}}$$

where $p^u_{\mathbf{P_1}}$ is the maximum approximation order among (1) the F2L translations performed for $\mathbf{P_1}$ and all of the ancestor nodes of $\mathbf{P_1}$ (denoted by $F2L(\mathbf{P_1})$); and (2) the direct local accumulations of $\mathbf{P_1}$ and those passed down from all of the ancestors of $\mathbf{P_1}$ (denoted by $DL(\mathbf{P_1})$). Shifting the expansion to another center $\mathbf{c}^*_{\mathbf{P_1}} \in \mathbf{P_1}$ is given by:

$$\widetilde{\Phi}(\mathbf{x}; F2L(\mathbf{P_1}) \cup DL(\mathbf{P_1}); \widetilde{N}(\mathbf{c}^*_{\mathbf{P_1}}, p^u_{\mathbf{P_1}})) \tag{7.2.7}$$

$$= \sum_{|\boldsymbol{\alpha}| \leq p^u_{\mathbf{P_1}}} \left[\sum_{\boldsymbol{\beta} \geq \boldsymbol{\alpha}} \binom{\boldsymbol{\beta}}{\boldsymbol{\alpha}} \widetilde{N}_{\boldsymbol{\beta}}(F2L(\mathbf{P_1}) \cup DL(\mathbf{P_1}), \mathbf{c_{P_1}})(\mathbf{c}^*_{\mathbf{P_1}} - \mathbf{c_{P_1}})^{\boldsymbol{\beta}-\boldsymbol{\alpha}} \right] (\mathbf{x} - \mathbf{c}^*_{\mathbf{P_1}})^{\boldsymbol{\alpha}}$$

$$= \sum_{|\boldsymbol{\alpha}| \leq p^u_{\mathbf{P_1}}} \widetilde{N}_{\boldsymbol{\alpha}}(F2L(\mathbf{P_1}) \cup DL(\mathbf{P_1}), \mathbf{c}^*_{\mathbf{P_1}})(\mathbf{x} - \mathbf{c}^*_{\mathbf{P_1}})^{\boldsymbol{\alpha}} \tag{7.2.8}$$

This shifted moments are added to the local moments of each child of $\mathbf{P_1}$, in effect transmitting the pruned contributions downward. At each query leaf, we evaluate the resulting local expansion at each query point.

7.2.2 Far-field Expansion for Three-body Potentials ($n = 3$)

In this section, we define far-field expansions for a three-body potential that is a product of functions of pairwise distances (see Equation (1.1.1)):

$$\phi(\mathbf{x_{i_1}}, \mathbf{x_{i_2}}, \mathbf{x_{i_3}}) = \phi_{1,2}(\mathbf{x_{i_1}}, \mathbf{x_{i_2}}) \cdot \phi_{1,3}(\mathbf{x_{i_1}}, \mathbf{x_{i_3}}) \cdot \phi_{2,3}(\mathbf{x_{i_2}}, \mathbf{x_{i_3}}) \tag{7.2.9}$$

We define the far-field moments of a node the same way defined for the pairwise potential case. Suppose we are given three nodes $\mathbf{P_1} \neq \mathbf{P_2} \neq \mathbf{P_3}$ from the tree. The following $(n-1)$-nested sum expresses the contribution for $x \in \mathbf{P_1}$ due to the other nodes $\mathbf{P_2}$ and $\mathbf{P_3}$:

$$\Phi(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}) = \sum_{\mathbf{x_{i_2}} \in \mathbf{P_2}} \sum_{\mathbf{x_{i_3}} \in \mathbf{P_3}} \phi(\mathbf{x}, \mathbf{x_{i_2}}, \mathbf{x_{i_3}}) \tag{7.2.10}$$

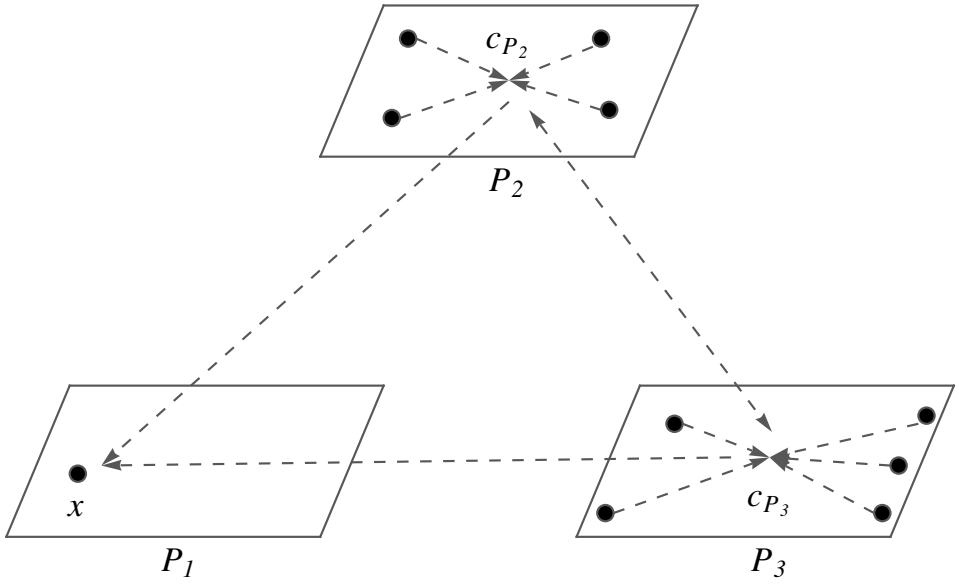


Figure 41: A far-field expansion at $\mathbf{x_{i_1}}$ created by the moments of $\mathbf{P_2}$ and $\mathbf{P_3}$. Note the double-arrow between the nodes $\mathbf{P_2}$ and $\mathbf{P_3}$ corresponding to the basis functions $D^{\boldsymbol{\alpha}-\boldsymbol{\alpha}_{1,2}-\boldsymbol{\alpha}_{1,3}}\phi_{2,3}(\mathbf{c_{P_2}} - \mathbf{c_{P_3}})$ (see Equation 7.2.11).

The basic goal here is to decompose Equation (7.2.10) into sums of products of the far-field moments of each node. A far-field expansion for $\mathbf{x_{i_1}} \in \mathbf{P_1}$ induced by the far-field moments of $\mathbf{P_2}$ and $\mathbf{P_3}$ is given by (see Figure 41):

$$
\begin{aligned}
&\Phi(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}) \\
&= \sum_{\mathbf{x_{i_2}} \in \mathbf{P_2}} \sum_{\mathbf{x_{i_3}} \in \mathbf{P_3}} \sum_{\boldsymbol{\alpha}_{1,2} \geq 0} \frac{(\mathbf{x_{i_2}} - \mathbf{c_{P_2}})^{\boldsymbol{\alpha}_{1,2}}}{\boldsymbol{\alpha}_{1,2}!} (-1)^{\boldsymbol{\alpha}_{1,2}} D^{\boldsymbol{\alpha}_{1,2}} \phi_{1,2}(\mathbf{x} - \mathbf{c_{P_2}}) \\
&\quad \sum_{\boldsymbol{\alpha}_{1,3} \geq 0} \frac{(\mathbf{x_{i_3}} - \mathbf{c_{P_3}})^{\boldsymbol{\alpha}_{1,3}}}{\boldsymbol{\alpha}_{1,3}!} (-1)^{\boldsymbol{\alpha}_{1,3}} D^{\boldsymbol{\alpha}_{1,3}} \phi_{1,3}(\mathbf{x} - \mathbf{c_{P_3}}) \\
&\quad \sum_{\boldsymbol{\alpha}_{2,3} \geq 0} \sum_{\boldsymbol{\beta}_{2,3} \leq \boldsymbol{\alpha}_{2,3}} \frac{(\mathbf{x_{i_2}} - \mathbf{c_{P_2}})^{\boldsymbol{\beta}_{2,3}}}{\boldsymbol{\beta}_{2,3}!} \frac{(\mathbf{x_{i_3}} - \mathbf{c_{P_3}})^{\boldsymbol{\alpha}_{2,3} - \boldsymbol{\beta}_{2,3}}}{(\boldsymbol{\alpha}_{2,3} - \boldsymbol{\beta}_{2,3})!} (-1)^{\boldsymbol{\alpha}_{2,3} - \boldsymbol{\beta}_{2,3}} D^{\boldsymbol{\alpha}_{2,3}} \phi_{2,3}(\mathbf{c_{P_2}} - \mathbf{c_{P_3}})
\end{aligned}
$$

By setting $\boldsymbol{\alpha} = \boldsymbol{\alpha}_{1,2} + \boldsymbol{\alpha}_{1,3} + \boldsymbol{\alpha}_{2,3}$ and pushing the summations over $\mathbf{x_{i_2}} \in \mathbf{P_2}$ and $\mathbf{x_{i_3}} \in \mathbf{P_3}$ inside, we get:

$$\begin{aligned}\Phi(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}) = &\sum_{\boldsymbol{\alpha} \geq 0} \sum_{\boldsymbol{\alpha}_{1,2} \leq \boldsymbol{\alpha}} \sum_{\boldsymbol{\alpha}_{1,3} \leq \boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2}} \sum_{\boldsymbol{\beta}_{2,3} \leq \boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\alpha}_{1,3}} \binom{\boldsymbol{\alpha}_{1,2} + \boldsymbol{\beta}_{2,3}}{\boldsymbol{\alpha}_{1,2}} \binom{\boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\beta}_{2,3}}{\boldsymbol{\alpha}_{1,3}} \\ &M_{\boldsymbol{\alpha}_{1,2} + \boldsymbol{\beta}_{2,3}}(\mathbf{P_2}, \mathbf{c_{P_2}}) M_{\boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\beta}_{2,3}}(\mathbf{P_3}, \mathbf{c_{P_3}}) (-1)^{\boldsymbol{\beta}_{2,3}} \\ &D^{\boldsymbol{\alpha}_{1,2}} \phi_{1,2}(\mathbf{x_{i_1}} - \mathbf{c_{P_2}}) D^{\boldsymbol{\alpha}_{1,3}} \phi_{1,3}(\mathbf{x_{i_1}} - \mathbf{c_{P_3}}) D^{\boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\alpha}_{1,3}} \phi_{2,3}(\mathbf{c_{P_2}} - \mathbf{c_{P_3}})\end{aligned} \tag{7.2.11}$$

Truncating $\boldsymbol{\alpha}$ at p-th order yields:

$$\begin{aligned}&\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}; \mathsf{F}(\mathbf{c_{P_2}} \times \mathbf{c_{P_3}}, p)) \\ =&\sum_{|\boldsymbol{\alpha}| \leq p} \sum_{\boldsymbol{\alpha}_{1,2} \leq \boldsymbol{\alpha}} \sum_{\boldsymbol{\alpha}_{1,3} \leq \boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2}} \sum_{\boldsymbol{\beta}_{2,3} \leq \boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\alpha}_{1,3}} \binom{\boldsymbol{\alpha}_{1,2} + \boldsymbol{\beta}_{2,3}}{\boldsymbol{\alpha}_{1,2}} \binom{\boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\beta}_{2,3}}{\boldsymbol{\alpha}_{1,3}} \\ &M_{\boldsymbol{\alpha}_{1,2} + \boldsymbol{\beta}_{2,3}}(\mathbf{P_2}, \mathbf{c_{P_2}}) M_{\boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\beta}_{2,3}}(\mathbf{P_3}, \mathbf{c_{P_3}}) (-1)^{\boldsymbol{\beta}_{2,3}} \\ &D^{\boldsymbol{\alpha}_{1,2}} \phi_{1,2}(\mathbf{x_{i_1}} - \mathbf{c_{P_2}}) D^{\boldsymbol{\alpha}_{1,3}} \phi_{1,3}(\mathbf{x_{i_1}} - \mathbf{c_{P_3}}) D^{\boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2} - \boldsymbol{\alpha}_{1,3}} \phi_{2,3}(\mathbf{c_{P_2}} - \mathbf{c_{P_3}})\end{aligned} \tag{7.2.12}$$

where $\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}; \mathsf{F}(\mathbf{c_{P_2}} \times \mathbf{c_{P_3}}, p))$ reads as "the p-th order far-field expansion at x due to the moments of $\mathbf{P_2}$ centered at $\mathbf{c_{P_2}}$ and the moments of $\mathbf{P_3}$ centered at $\mathbf{c_{P_3}}$."

Computational Cost of Evaluating the Far-field Expansion. The first three summations over $\boldsymbol{\alpha}$, $\boldsymbol{\alpha}_{1,2}$, $\boldsymbol{\alpha}_{1,3}$ collectively contribute $\mathcal{O}(p^3)$ terms, and the inner summation contributing at most $\mathcal{O}(p^3)$ terms. Thus, evaluating the p-th order far-field expansion for a three-body potential on a single point takes $\mathcal{O}(p^6)$ time.

7.2.3 Far-field Expansion for General Multibody Potentials ($n \geq 2$)

For a general multibody potential that can be expressed as products of pairwise functions (see Equation (7.0.3)), the far-field expansion induced by the points in $\mathbf{P_2}, \cdots, \mathbf{P_n}$ for $\mathbf{x} \in \mathbf{P_1}$ is:

$$\begin{aligned}&\Phi(\mathbf{x}; \mathbf{P_2} \times \cdots \times \mathbf{P_n}) \\ =&\prod_{2 \leq k \leq n} \sum_{\mathbf{x_{i_k}} \in \mathbf{P_k}} \sum_{\boldsymbol{\alpha}_{1,k} \geq 0} \frac{(\mathbf{x_{i_k}} - \mathbf{c_{P_k}})^{\boldsymbol{\alpha}_{1,k}}}{\boldsymbol{\alpha}_{1,k}!} (-1)^{\boldsymbol{\alpha}_{1,k}} D^{\boldsymbol{\alpha}_{1,k}} \phi_{1,k}(\mathbf{x} - \mathbf{c_{P_k}}) \\ &\prod_{2 \leq s < t \leq n} \sum_{\boldsymbol{\alpha}_{s,t} \geq 0} \sum_{\boldsymbol{\beta}_{s,t} \leq \boldsymbol{\alpha}_{s,t}} \frac{(\mathbf{x_{i_s}} - \mathbf{c_{P_s}})^{\boldsymbol{\beta}_{s,t}}}{\boldsymbol{\beta}_{s,t}!} \frac{(x_{i_t} - \mathbf{c_{P_t}})^{\boldsymbol{\alpha}_{s,t} - \boldsymbol{\beta}_{s,t}}}{(\boldsymbol{\alpha}_{s,t} - \boldsymbol{\beta}_{s,t})!} (-1)^{\boldsymbol{\alpha}_{s,t} - \boldsymbol{\beta}_{s,t}} D^{\boldsymbol{\alpha}_{s,t}} \phi_{s,t}(\mathbf{c_{P_s}} - \mathbf{c_{P_t}})\end{aligned}$$

Focus on grouping and multiplying monomial powers of $(\mathbf{x}_{i_k} - \mathbf{c}_{\mathbf{P_k}})$ for each $2 \leq k \leq n$:

$$\frac{(\mathbf{x}_{i_k} - \mathbf{c}_{\mathbf{P_k}})^{\boldsymbol{\alpha}_{1,k} + \sum_{u=2}^{k-1}(\boldsymbol{\alpha}_{u,k} - \boldsymbol{\beta}_{u,k}) + \sum_{v=k+1}^{n} \boldsymbol{\beta}_{k,v}}}{\boldsymbol{\alpha}_{1,k}! \prod_{u=2}^{k-1}(\boldsymbol{\alpha}_{u,k} - \boldsymbol{\beta}_{u,k})! \prod_{v=k+1}^{n} \boldsymbol{\beta}_{k,v}!}$$

Let $\boldsymbol{\xi}_k = \boldsymbol{\alpha}_{1,k} + \sum_{u=2}^{k-1}(\boldsymbol{\alpha}_{u,k} - \boldsymbol{\beta}_{u,k}) + \sum_{v=k+1}^{n} \boldsymbol{\beta}_{k,v}$ and $b_k = \frac{\boldsymbol{\xi}_k!}{\boldsymbol{\alpha}_{1,k}! \prod_{u=2}^{k-1}(\boldsymbol{\alpha}_{u,k} - \boldsymbol{\beta}_{u,k})! \prod_{v=k+1}^{n} \boldsymbol{\beta}_{k,v}!}$. Then,

$$\begin{aligned}
&\Phi(\mathbf{x}; \mathbf{P_2} \times \cdots \times \mathbf{P_n}) \\
&= \prod_{2 \leq s < t \leq n} \prod_{2 \leq k \leq n} \sum_{\boldsymbol{\alpha}_{1,k} \geq 0} \sum_{\boldsymbol{\alpha}_{s,t} \geq 0} \sum_{\boldsymbol{\beta}_{s,t} \leq \boldsymbol{\alpha}_{s,t}} b_k \; M_{\boldsymbol{\xi}_k}(\mathbf{P_k}, \mathbf{c}_{\mathbf{P_k}}) \; (-1)^{\boldsymbol{\beta}_{s,t}} \; D^{\boldsymbol{\alpha}_{1,k}} \phi_{1,k}(\mathbf{x} - \mathbf{c}_{\mathbf{P_k}}) D^{\boldsymbol{\alpha}_{s,t}} \phi_{s,t}(\mathbf{c}_{\mathbf{P_s}} - \mathbf{c}_{\mathbf{P_t}})
\end{aligned} \tag{7.2.13}$$

Equation (7.2.13) is a convolution of far-field moments of $\mathbf{P_2}, \cdots, \mathbf{P_n}$. We can truncate the expansion above for terms for $|\boldsymbol{\alpha}| = \left| \sum_{1 \leq r < s \leq n} \boldsymbol{\alpha}_{r,s} \right| > p$ for some $p > 0$. Note that Equation (7.2.13) includes the $n = 2$ and $n = 3$ cases.

$$\begin{aligned}
&\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2} \times \cdots \times \mathbf{P_n}; \mathsf{F}(\mathbf{c}_{\mathbf{P_2}} \times \cdots \times \mathbf{c}_{\mathbf{P_n}}, p)) \\
&= \prod_{2 \leq s < t \leq n} \prod_{2 \leq k \leq n} \sum_{|\boldsymbol{\alpha}| \leq p} \sum_{\boldsymbol{\alpha}_{1,k} \geq 0} \sum_{\boldsymbol{\alpha}_{s,t} \geq 0} \sum_{\boldsymbol{\beta}_{s,t} \leq \boldsymbol{\alpha}_{s,t}} b_k \; M_{\boldsymbol{\xi}_k}(\mathbf{P_k}, \mathbf{c}_{\mathbf{P_k}}) \; (-1)^{\boldsymbol{\beta}_{s,t}} \\
&D^{\boldsymbol{\alpha}_{1,k}} \phi_{1,k}(\mathbf{x} - \mathbf{c}_{\mathbf{P_k}}) D^{\boldsymbol{\alpha}_{s,t}} \phi_{s,t}(\mathbf{c}_{\mathbf{P_s}} - \mathbf{c}_{\mathbf{P_t}})
\end{aligned} \tag{7.2.14}$$

Computational Cost of Evaluating the Far-field Expansion. The summations over $\boldsymbol{\alpha}_{r,s}$ for $1 \leq r < s \leq n$ collectively contribute $\mathcal{O}(p^3)$ terms, and each inner summation over $\boldsymbol{\beta}_{s,t}$ contributing at most $\mathcal{O}(p^3)$ terms. Thus, evaluating the p-th order far-field expansion for a general multibody potential of the form Equation (7.0.3) on a single point takes $\mathcal{O}\left(p^{3\binom{n-1}{2}+1)}\right)$ time. In practice, we are forced to use $p = 0$ for $n > 2$ unless most $\phi_{p,q}(\mathbf{x}_{\mathbf{i_p}}, \mathbf{x}_{\mathbf{i_q}})$'s in Equation (7.0.3) are constant functions.

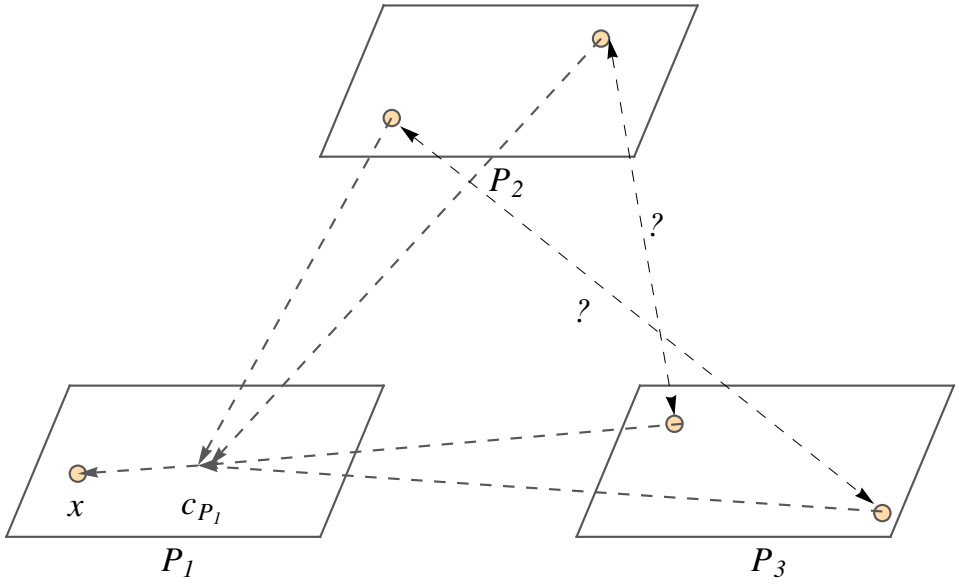


Figure 42: A local expansion created inside the node $\mathbf{P_1}$ at x by directly accumulating each point in $\mathbf{P_2}$ and $\mathbf{P_3}$ (see Equation (7.2.15)). We are not aware of a technique to express an interaction between a particle in $\mathbf{P_2}$ and a particle in $\mathbf{P_3}$ (marked by the ? symbol) for $p > 0$.

7.2.4 Local Expansion for Three-body Potentials ($n = 3$)

Unlike the far-field expansion case, we are presented a fundamental difficulty. In order to derive a local expansion, we need to express the influence of each non-evaluation point $\mathbf{x_{i_j}}$ on the evaluation point $\mathbf{x}$ at a center near $\mathbf{x}$. However, breaking up the interaction among the non-evaluation points (i.e. $\mathbf{x_{i_j}}$'s in the arguments of $\phi(\mathbf{x}, \mathbf{x_{i_1}}, \cdots, \mathbf{x_{i_{n-1}}})$) without loss of information is hard. To see this: take a three-body potential expressible in products of pairwise functions (see Figure 42). Expanding near $\mathbf{c_{P_1}}$ inside the node $\mathbf{P_1}$ yields an expansion valid for $\mathbf{x} \in \mathbf{P_1}$:

$$\begin{aligned}
&\Phi(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}) \\
&= \sum_{\mathbf{x_{i_2}} \in \mathbf{P_2}} \sum_{\mathbf{x_{i_3}} \in \mathbf{P_3}} \sum_{\boldsymbol{\alpha}_{1,2} \geq 0} \frac{D^{\boldsymbol{\alpha}_{1,2}} \phi_{1,2}(\mathbf{c_{P_1}} - \mathbf{x_{i_2}})}{\boldsymbol{\alpha}_{1,2}!} (\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\alpha}_{1,2}} \sum_{\boldsymbol{\alpha}_{1,3} \geq 0} \frac{D^{\boldsymbol{\alpha}_{1,3}} \phi_{1,3}(\mathbf{c_{P_1}} - \mathbf{x_{i_3}})}{\boldsymbol{\alpha}_{1,3}!} (\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\alpha}_{1,3}} \\
&\quad \sum_{\boldsymbol{\alpha}_{2,3} \geq 0} \frac{D^{\boldsymbol{\alpha}_{2,3}} \phi_{2,3}(\mathbf{c_{P_1}} - \mathbf{x_{i_3}})}{\boldsymbol{\alpha}_{2,3}!} (\mathbf{x_{i_2}} - \mathbf{c_{P_1}})^{\boldsymbol{\alpha}_{2,3}}
\end{aligned}$$

Again, let $\boldsymbol{\alpha} = \boldsymbol{\alpha}_{1,2} + \boldsymbol{\alpha}_{1,3} + \boldsymbol{\alpha}_{2,3}$. Switching the orders of summations results:

$$\begin{aligned}\Phi(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}) = &\sum_{\boldsymbol{\alpha} \geq 0} \sum_{\boldsymbol{\alpha}_{1,2} \leq \boldsymbol{\alpha}} \sum_{\boldsymbol{\alpha}_{1,3} \leq \boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2}} \left[\sum_{\mathbf{x_{i_2}} \in \mathbf{P_2}} \frac{D^{\boldsymbol{\alpha}_{1,2}} \phi_{1,2}(\mathbf{c_{P_1}} - \mathbf{x_{i_2}})}{\boldsymbol{\alpha}_{1,2}!} (\mathbf{x_{i_2}} - \mathbf{c_{P_1}})^{\boldsymbol{\alpha}_{2,3}} \right] \\ &\left[\sum_{\mathbf{x_{i_3}} \in \mathbf{P_3}} \frac{D^{\boldsymbol{\alpha}_{1,3}} \phi_{1,3}(\mathbf{c_{P_1}} - \mathbf{x_{i_3}})}{\boldsymbol{\alpha}_{1,3}!} \frac{D^{\boldsymbol{\alpha}_{2,3}} \phi_{2,3}(\mathbf{c_{P_1}} - \mathbf{x_{i_3}})}{\boldsymbol{\alpha}_{2,3}!} \right] (\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\alpha}_{1,2} + \boldsymbol{\alpha}_{1,3}} \\ = &\sum_{\boldsymbol{\alpha} \geq 0} \left[\sum_{\boldsymbol{\alpha}_{1,2} \leq \boldsymbol{\alpha}} \sum_{\boldsymbol{\alpha}_{1,3} \leq \boldsymbol{\alpha} - \boldsymbol{\alpha}_{1,2}} \bar{N}_{\boldsymbol{\alpha}}(\mathbf{P_2}, \mathbf{c_{P_1}}) \; \bar{N}_{\boldsymbol{\alpha}}(\mathbf{P_3}, \mathbf{c_{P_1}}) \right] (\mathbf{x} - \mathbf{c_{P_1}})^{\boldsymbol{\alpha}_{1,2} + \boldsymbol{\alpha}_{1,3}}\end{aligned} \tag{7.2.15}$$

We need the exponent of $(\mathbf{x} - \mathbf{c_{P_1}})$ to match $\boldsymbol{\alpha}$ to be able to define the local moments inside $\mathbf{P_1}$. Unless $\boldsymbol{\alpha}_{2,3} = 0$ (i.e. ignore the interaction between a particle in the second set and a particle in the third set), this is not possible. Since we encounter a similar problem in the general case, we will skip its discussion.

7.3 Simpler Algorithm for General Multibody Potentials

Instead of trying to derive the full-fledged tools for general multibody potentials, we focus on deriving something simpler. Let us focus on the $n = 3$ case. For a given set of three pairwise disjoint nodes: $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ and a monotonically decreasing[3] three-body potentials such as $\phi(\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}) = \frac{1}{||\mathbf{x_1} - \mathbf{x_2}||^{\nu_{1,2}} ||\mathbf{x_1} - \mathbf{x_3}||^{\nu_{1,3}} ||\mathbf{x_2} - \mathbf{x_3}||^{\nu_{2,3}}}$,

$$\begin{aligned}\forall \mathbf{x_i} \in \mathbf{P_1}, \widetilde{\Phi}(\mathbf{x_i}; \mathbf{P_2} \times \mathbf{P_3}) &= |\mathbf{P_2}||\mathbf{P_3}| \phi(\mathbf{c_{P_1}}, \mathbf{c_{P_2}}, \mathbf{c_{P_3}}) \\ \forall \mathbf{x_j} \in \mathbf{P_2}, \widetilde{\Phi}(\mathbf{x_j}; \mathbf{P_1} \times \mathbf{P_3}) &= |\mathbf{P_1}||\mathbf{P_3}| \phi(\mathbf{c_{P_1}}, \mathbf{c_{P_2}}, \mathbf{c_{P_3}}) \\ \forall \mathbf{x_k} \in \mathbf{P_3}, \widetilde{\Phi}(\mathbf{x_k}; \mathbf{P_1} \times \mathbf{P_2}) &= |\mathbf{P_1}||\mathbf{P_2}| \phi(\mathbf{c_{P_1}}, \mathbf{c_{P_2}}, \mathbf{c_{P_3}})\end{aligned}$$

which can be obtained by setting $p = 0$ in Equation (7.2.12). This means that we can get a cheaper approximation using the number of points owned by each node. Using the pairwise minimum and maximum node distances yields:

$$\phi(d^u(\mathbf{P_1}, \mathbf{P_2}), d^u(\mathbf{P_1}, \mathbf{P_3}), d^u(\mathbf{P_2}, \mathbf{P_3})) \leq \phi(\mathbf{c_{P_1}}, \mathbf{c_{P_2}}, \mathbf{c_{P_3}}) \leq \phi(d^l(\mathbf{P_1}, \mathbf{P_2}), d^l(\mathbf{P_1}, \mathbf{P_3}), d^l(\mathbf{P_2}, \mathbf{P_3}))$$

[3] "Monotonic" multibody potentials decrease in value if one of the Euclidean distance arguments is increased while the other two are held constant.

It is straightforward to generalize this for the $n \geq 2$ case.

Non-monotonic Potentials: For non-monotonic potentials such as the Lennard-Jones potential $\phi(\mathbf{x_1}, \mathbf{x_2}) = \frac{a}{r^{12}} - \frac{b}{r^6}$, we can compute the critical points of ϕ and determine the intervals of monotonicity of ϕ and consider how ϕ behaves in the distance bound range between $d^l(\mathbf{P_1}, \mathbf{P_2})$ and $d^u(\mathbf{P_1}, \mathbf{P_2})$. We take a simpler approach that results in an algorithm that is easier to code; we break up the potential into two parts such that $\phi(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n}) = \phi^+(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n}) - \phi^-(\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_n})$, and get a lower and upper bound (though a looser bound) on the contributions from the positive potential ϕ^+ and negative potential ϕ^-.

7.3.1 Specifying the Approximation Rules

The overall algorithm which also subsumes the pairwise potential case ($n = 2$) was shown in Algorithm 7.2.1. We can now specify the CanSummarize function for the general multibody case. For guaranteeing τ absolute error bound criterion (Definition 2.4.1), the CanSummarize function returns true if:

$$\left|\phi(d^u(\mathbf{P_1}, \mathbf{P_2}), \cdots, d^u(\mathbf{P_{n-1}}, \mathbf{P_n})) - \phi(d^l(\mathbf{P_1}, \mathbf{P_2}), \cdots, d^l(\mathbf{P_{n-1}}, \mathbf{P_n}))\right| \leq \frac{\tau}{T^{root}}$$

where $T^{root} = \binom{N-1}{n-1}$ (i.e. the total number of tuples in each slice in Figure 39). Let us also define T_i to be the number of tuples containing a fixed particle in $\mathbf{P_i}$ (see Figure 40). For example, for $n = 3$, the corresponding Summarize function would accumulate for each node:

for $\mathbf{P_1}$: $|\mathbf{P_2}||\mathbf{P_3}|\phi(\mathbf{c_{P_1}}, \mathbf{c_{P_2}}, \mathbf{c_{P_3}})$, for $\mathbf{P_2}$: $|\mathbf{P_1}||\mathbf{P_3}|\phi(\mathbf{c_{P_1}}, \mathbf{c_{P_2}}, \mathbf{c_{P_3}})$, and for $\mathbf{P_3}$: $|\mathbf{P_1}||\mathbf{P_2}|\phi(\mathbf{c_{P_1}}, \mathbf{c_{P_2}}, \mathbf{c_{P_3}})$.

Hybrid Absolute/Relative Error Guarantee. The algorithm for guaranteeing the hybrid absolute/relative error bound (Definition 2.4.2) deterministically ($\alpha = 0$) is not so much different from that for guaranteeing the absolute error bound. In each node P, we maintain the lower bound on the accumulated potentials for the particles in P (denoted as $\Phi^l(P)$, a *summary statistic* stored in P). The function

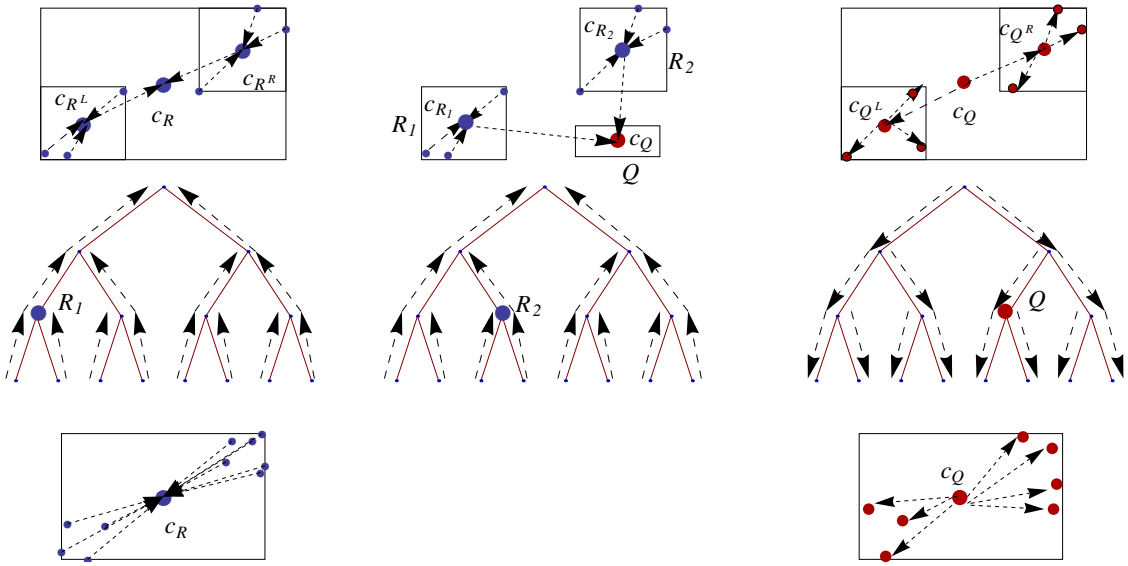


Figure 43: Three-body multipole methods for $p = 0$ in a nutshell.

CanSummarize returns true if,

$$\left|\phi(d^u(\mathbf{P_1}, \mathbf{P_2}), \cdots, d^u(\mathbf{P_{n-1}}, \mathbf{P_n})) - \phi(d^l(\mathbf{P_1}, \mathbf{P_2}), \cdots, d^l(\mathbf{P_{n-1}}, \mathbf{P_n}))\right| \leq \frac{\epsilon \min_{1 \leq i \leq n} (\Phi^l(\mathbf{P_i}) + \delta^l(\mathbf{P_i}; \mathbf{P_1} \times \cdots \times \mathbf{P_{i-1}} \times \mathbf{P_{i+1}} \times \cdots \times \mathbf{P_n})) + \tau}{T^{root}} \tag{7.3.1}$$

where each $\delta^l(\mathbf{P_i}; \mathbf{P_1} \times \cdots \times \mathbf{P_{i-1}} \times \mathbf{P_{i+1}} \times \cdots \times \mathbf{P_n}) = \prod_{\substack{1 \leq j \leq i-1 \\ i+1 \leq j \leq n}} |\mathbf{P_j}| \phi(d^u(\mathbf{P_1}, \mathbf{P_2}), \cdots, d^u(\mathbf{P_{n-1}}, \mathbf{P_n}))$ (which is computed just using the contribution of the other nodes on the i-th node) is added to the currently running lower bound on each node $\Phi^l(\mathbf{P_i})$ to reflect the most recently available information on the lower bound. $\Phi^l(\mathbf{P_i})$ can be incremented and tightened as the computation progresses, either in the base case or when the recursive sub-calls in Algorithm 7.2.1 are completed (Line 11).

Monte Carlo-based Approximations. The error bounds provided by the bounding boxes (see Figure 10) assume that all pairs of points selected between the two nodes are collapsed to two positions that achieve the minimum distance (and vice versa for the maximum distance); therefore, these bounds are very pessimistic and loose. Here we introduce a method for approximating the potential sums with a

Algorithm 7.3.1 CANSUMMARIZE($\{\mathbf{P_i}\}_{i=1}^n$): the Monte Carlo-based approximation.

```
if ζ · m_limit ≤ min{T_1, T_2, T_3} then
  for each P_i ∈ {P_i}_{i=1}^n do
    if i == 1 or P_i ≠ P_{i−1} then
      for x_i ∈ P_i do
        if CANSUMMARIZEMCPOINT(x_i, i, {P_i}_{i=1}^n) == false then
          return false
  return true
else
  return false
```

probabilistic bound satisfying Definition 2.4.3. We can trade determinism for further gain in efficiency. We have an additional parameter α that controls the probability level at which the deviation between each approximation and its corresponding exact values holds. This was introduced first in [96, 95] for probabilistic approximations of aggregate sums and later extended in [113] to handle per-particle quantities. The theorem that we rely on for probabilistic approximation is the following:

For three-body potentials, suppose we are given the set of three nodes, $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$. Let us consider $\mathbf{x} \in \mathbf{P_1}$ (similar approximations can be made for each point in $\mathbf{P_2}$ and $\mathbf{P_3}$), and the contribution of $\mathbf{P_2}$ and $\mathbf{P_3}$ to its potential sum:

$$\Phi(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}) = \sum_{\mathbf{x_{i_2}} \in \mathbf{X} \setminus \{\mathbf{x}\}} \sum_{\substack{\mathbf{x_{i_3}} \in \mathbf{X} \setminus \{\mathbf{x}\} \\ i_2 < i_3}} \phi\left(\mathbf{x}, \mathbf{x_{i_2}}, \mathbf{x_{i_3}}\right)$$

We can sample m potential values $\phi(\mathbf{x_{i_1}}, \mathbf{x_{i_2}}, \mathbf{x_{i_3}})$ from the empirical distribution F formed by the 3-tuples formed among S_1, S_2, and S_3 that contain x in the list. From the m samples, we get the empirical distribution F_m^x, from which we form an approximate $\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3})$:

$$\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}; F_m^{\mathbf{x}}) = T_1 \widetilde{\mu}_{F_m^{\mathbf{x_i}}} = \frac{T_1}{m} \sum_{s=1}^{m} \phi(\mathbf{x_{i_1}^s}, \mathbf{x_{i_2}^s}, \mathbf{x_{i_3}^s})$$

where $\mathbf{x_{i_1}^s} = \mathbf{x}$ for all $1 \leq s \leq m$. For sufficiently large values of m, we can assume that the discrepancy provided by the Berry-Esseen theorem is small and concentrate on the sample variance of the sample mean distribution. The sample variance of the sample mean distribution $\widetilde{\sigma}_{\mu_{F_m^{\mathbf{x_i}}}}$ is given by:

Algorithm 7.3.2 CANSUMMARIZEMCPOINT($\mathbf{x}, i, \{\mathbf{S_i}\}_{i=1}^n$): Pruning function for the Monte Carlo based approximation per each point.

$F^{\mathbf{x}} \leftarrow \emptyset$
repeat
  Get a random n-tuple $(\mathbf{x_1}, \cdots, \mathbf{x_{i-1}}, \mathbf{x}, \mathbf{x_{i+1}}, \cdots, \mathbf{x_n})$ where $\mathbf{x_j} \in \mathbf{S_j}$
  $F^{\mathbf{x}} \leftarrow F^{\mathbf{x}} \cup \{\phi(\mathbf{x}_j, \cdots, \mathbf{x_{j-1}}, \mathbf{x}, \mathbf{x_{j+1}}, \cdots, \mathbf{x_n})\}$
until ($z_{\alpha/2}\widetilde{\sigma}_{\mu_{F^x}} \leq \frac{\tau}{T^{root}}$ and $|F^{\mathbf{x}}| \geq 30$) or $|F^{\mathbf{x}}| \geq m_{limit}$
return $z_{\alpha/2}\widetilde{\sigma}_{\mu_{F^{\mathbf{x}}}} \leq \frac{\tau}{T^{root}}$

$$\widetilde{\sigma}_{\mu_{F_m^{\mathbf{x_i}}}} = \frac{\widetilde{\sigma}_{F_m^{\mathbf{x_i}}}}{\sqrt{m}} = \frac{1}{\sqrt{m}}\sqrt{\frac{1}{m-1}\sum_{s=1}^{m}(\phi(\mathbf{x_{i_1^s}}, \mathbf{x_{i_2^s}}, \mathbf{x_{i_3^s}}) - \widetilde{\mu}_{F_m^{\mathbf{x_i}}})^2}$$

where $\widetilde{\sigma}$ is the sample variance. Given m i.i.d. samples, with probability of at least $(1-\alpha)$,

$$\left|\widetilde{\Phi}(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3}; F_m^{\mathbf{x}}) - \Phi(\mathbf{x}; \mathbf{P_2} \times \mathbf{P_3})\right| \leq T_1 z_{\alpha/2}\widetilde{\sigma}_{\mu_{F_m^{\mathbf{x_i}}}}$$

where $z_{\alpha/2}$ is the number of standard deviations on either side of $\widetilde{\mu}_{F_m^{\mathbf{x_i}}}$ to give at least $(1-\alpha)$ coverage under the normal distribution.

Modifications to the algorithm. A Monte Carlo sampling based routine is shown in Algorithm 7.3.1. The function CANSUMMARIZE determines whether performing Monte Carlo approximations (which involves iterating over each unique point $\mathbf{x} \in \bigcup_{i=1}^{n} \mathbf{P_i}$) with at least m_{limit} samples is computationally cheaper than the brute-force computation. ζ is a global variable that dictates the desired amount of speedup needed for applying Monte Carlo approximations, rather than recursing to smaller subsets of the three nodes. If a desired speedup could be achieved, it loops for each unique point in $\mathbf{x} \in \bigcup_{i=1}^{n} \mathbf{P_i}$ and computes the sample mean of the potential values of the tuples that contain $\mathbf{x}$, and the corresponding variance of the sample mean until (1) the desired error is achieved; or (2) exceeds the number of trial samples m_{limit}. Algorithm 7.3.1 is the form used for bounding the absolute error of each potential sum error by τ with at least probability of $(1-\alpha)$. For bounding the hybrid absolute/relative error with at least probability of $(1-\alpha)$ (Definition 2.4.3), we replace the termination condition

Algorithm 7.3.3 SUMMARIZEMC($\{\mathbf{S_i}\}_{i=1}^n, \{T_i\}_{i=1}^n, \beta$): Monte Carlo based approximation.

for each $\mathbf{S_i} \in \{\mathbf{S_i}\}_{i=1}^n$ **do**
  if $i == 1$ or $\mathbf{S_i} \neq \mathbf{S_{i-1}}$ **then**
    for $\mathbf{x_i} \in \mathbf{S_i}$ **do**
      $\widetilde{\Phi}(\mathbf{x_i}) \leftarrow \widetilde{\Phi}(\mathbf{x_i}) + T_i \cdot \widetilde{\mu}_{F^{\mathbf{x_i}}}$, $\Phi^l(\mathbf{x_i}) \leftarrow \Phi^l(\mathbf{x_i}) + T_i \cdot \left(\widetilde{\mu}_{F^{\mathbf{x_i}}} - z_{\beta/2}\widetilde{\sigma}_{\mu_{F^{\mathbf{x_i}}}}\right)$

in the loop: $z_{\alpha/2}\widetilde{\sigma}_{\mu_{F^{\mathbf{x}}}} \leq \frac{\tau}{T^{root}}$ with:

$$T_i \cdot z_{\alpha/2}\widetilde{\sigma}_{\mu_{F^{\mathbf{x_i}}}} \leq \frac{\epsilon(\Phi^l(\mathbf{P_i}) + T_i(\widetilde{\mu}_{F^{\mathbf{x_i}}} - z_{\alpha/2}\widetilde{\sigma}_{\mu_{F^{\mathbf{x_i}}}})) + \tau T_i}{T^{root}} \tag{7.3.2}$$

7.4 Correctness of the Algorithm

The correctness of our algorithm for the deterministic hybrid absolute/relative error criterion is given by:

Theorem 7.4.1. *Algorithm 7.2.1 with the function* CANSUMMARIZE *with the relative error bound guarantee (Equation 7.3.1) produces approximation* $\widetilde{\Phi}(\mathbf{x_{i_1}})$ *for* $\mathbf{x_{i_1}} \in X$ *such that*

$$|\widetilde{\Phi}(\mathbf{x_{i_1}}) - \Phi(\mathbf{x_{i_1}})| \leq \epsilon\Phi(\mathbf{x_{i_1}}) + \tau \tag{7.4.1}$$

Proof. (By mathematical induction) For simplicity, let us focus on $n = 3$. We induct on the number of points $|\mathbf{P_1} \cup \mathbf{P_2} \cup \mathbf{P_3}|$ encountered during the recursion of the algorithm.

Base case: There are two parts to this part of the proof.

- Line 1 of the function MTPOTENTIALCANONICAL in Algorithm 7.2.1: any set of nodes $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ for which the function CANSUMMARIZE returns true

satisfies the error bounds for $\mathbf{x_{i_u}} \in \mathbf{P_u}$ for $u = 1, 2, 3$:

$$\begin{aligned}
\forall \mathbf{x_{i_1}} \in \mathbf{P_1}, &\left|\widetilde{\Phi}(\mathbf{x_{i_1}}; \mathbf{P_2} \times \mathbf{P_3}) - \Phi(\mathbf{x_{i_1}}; \mathbf{P_2} \times \mathbf{P_3})\right| \\
&\leq \frac{T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}} \left(\epsilon \Phi^l(\mathbf{P_1}) + \tau\right) \leq \frac{T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}} \left(\epsilon \Phi(\mathbf{x_{i_1}}) + \tau\right) \\
\forall \mathbf{x_{i_2}} \in \mathbf{P_2}, &\left|\widetilde{\Phi}(\mathbf{x_{i_2}}; \mathbf{P_1} \times \mathbf{P_3}) - \Phi(\mathbf{x_{i_2}}; \mathbf{P_1} \times \mathbf{P_3})\right| \\
&\leq \frac{T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}} \left(\epsilon \Phi^l(\mathbf{P_2}) + \tau\right) \leq \frac{T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}} \left(\epsilon \Phi(\mathbf{x_{i_2}}) + \tau\right) \\
\forall \mathbf{x_{i_3}} \in \mathbf{P_3}, &\left|\widetilde{\Phi}(\mathbf{x_{i_3}}; \mathbf{P_1} \times \mathbf{P_2}) - \Phi(\mathbf{x_{i_3}}; \mathbf{P_1} \times \mathbf{P_2})\right| \\
&\leq \frac{T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}} \left(\epsilon \Phi^l(\mathbf{P_3}) + \tau\right) \leq \frac{T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}} \left(\epsilon \Phi(\mathbf{x_{i_3}}) + \tau\right) \qquad (7.4.2)
\end{aligned}$$

where $T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}$ denotes the number of tuples chosen by fixing $\mathbf{x_{i_u}}$ and selecting the other two from $\mathbf{P_2}$ and $\mathbf{P_3}$ and so on.

- The function call MTPotentialBase in Algorithm 7.2.1: each $\mathbf{x_{i_1}} \in \mathbf{P_1}$ and $\mathbf{x_{i_2}} \in \mathbf{P_2}$ and $\mathbf{x_{i_3}} \in \mathbf{P_3}$ exchange contributions exactly and incur no approximation error.

Inductive step: Suppose we are given the set of three nodes $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$ (at least one of which is an internal node) in the function MTPotentialCanonical. Suppose the three tuples $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ could not be pruned, and that we need to recurse on each child of $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$.

By assumption, CanSummarize returns false if any one of the nodes $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ includes one of the other nodes (see Section 7.2). For $n = 3$, we can assume that the possible node tuple cases that could be considered for pruning are shown in Figure 40. Let $\{\{\mathbf{P_s^k}\}_{s=1}^3\}_{k=1}^t$ be the set of set of three nodes considered during the recursive sub-computations using the child nodes of each $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$; note that the maximum value of t is 8 for three-body interactions. Note that for each k, P_s^k is either (1) the node $\mathbf{P_s}$ itself (2) the left child node of $\mathbf{P_s}$ (3) the right child node of $\mathbf{P_s}$. Therefore, for each $k = 1, 2, \cdots, t$, $|\mathbf{P_1^k} \cup \mathbf{P_2^k} \cup \mathbf{P_3^k}| \leq |\mathbf{P_1} \cup \mathbf{P_2} \cup \mathbf{P_3}|$. The equality holds when all of $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$ are leaf nodes for which the error criterion is satisfied by the base case function (no error incurred).

If any one of $\mathbf{P_1}$, $\mathbf{P_2}$, and $\mathbf{P_3}$ is an internal node, then we are guaranteed that $|\mathbf{P_1^k} \cup \mathbf{P_2^k} \cup \mathbf{P_3^k}| < |\mathbf{P_1} \cup \mathbf{P_2} \cup \mathbf{P_3}|$ for all $k = 1, \cdots, t$. We invoke the inductive hypothesis to conclude that for each k and for each $\mathbf{x_{i_u}} \in \mathbf{P_u^k}$ for $u = 1, 2, 3$:

$$\begin{aligned}
\forall \mathbf{x_{i_1}} \in \mathbf{P_1^k}, &\left|\widetilde{\Phi}(\mathbf{x_{i_1}}; \mathbf{P_2^k} \times \mathbf{P_3^k}) - \Phi(\mathbf{x_{i_1}}; \mathbf{P_2^k} \times \mathbf{P_3^k})\right| \\
&\leq \frac{T_{\mathbf{x_{i_1}} \times \mathbf{P_2^k} \times \mathbf{P_3^k}}}{T^{root}} \left(\epsilon \Phi^l(\mathbf{P_1^k}) + \tau\right) \leq \frac{T_{\mathbf{x_{i_1}} \times \mathbf{P_2^k} \times \mathbf{P_3^k}}}{T^{root}} \left(\epsilon \Phi(\mathbf{x_{i_1}}) + \tau\right) \\
\forall \mathbf{x_{i_2}} \in \mathbf{P_2^k}, &\left|\widetilde{\Phi}(\mathbf{x_{i_2}}; \mathbf{P_1^k} \times \mathbf{P_3^k}) - \Phi(\mathbf{x_{i_2}}; \mathbf{P_1^k} \times \mathbf{P_3^k})\right| \\
&\leq \frac{T_{\mathbf{x_{i_2}} \times \mathbf{P_1^k} \times \mathbf{P_3^k}}}{T^{root}} \left(\epsilon \Phi^l(\mathbf{P_2^k}) + \tau\right) \leq \frac{T_{\mathbf{x_{i_2}} \times \mathbf{P_1^k} \times \mathbf{P_3^k}}}{T^{root}} \left(\epsilon \Phi(\mathbf{x_{i_2}}) + \tau\right) \\
\forall \mathbf{x_{i_3}} \in \mathbf{P_3^k}, &\left|\widetilde{\Phi}(\mathbf{x_{i_3}}; \mathbf{P_1^k} \times \mathbf{P_2^k}) - \Phi(\mathbf{x_{i_3}}; \mathbf{P_1^k} \times \mathbf{P_2^k})\right| \\
&\leq \frac{T_{\mathbf{x_{i_3}} \times \mathbf{P_1^k} \times \mathbf{P_2^k}}}{T^{root}} \left(\epsilon \Phi^l(\mathbf{P_3^k}) + \tau\right) \leq \frac{T_{\mathbf{x_{i_3}} \times \mathbf{P_1^k} \times \mathbf{P_2^k}}}{T^{root}} \left(\epsilon \Phi(\mathbf{x_{i_3}}) + \tau\right)
\end{aligned}$$

where T_s^k is the number of 3-tuples formed among P_1^k, P_2^k, P_3^k that contain a fixed point in P_s^k. By the triangle inequality, Equation 7.4.1 holds by extending to $\mathbf{P_1} = \mathbf{P_2} = \mathbf{P_3} = X$ since the number of encountered tuples for each particle add up to T^{root}. □

We are now ready to prove the correctness of our algorithm for bounding the relative error probabilistically.

Theorem 7.4.2. *Algorithm 7.2.1 with the function* CanSummarize *with the modification described in Equation 7.3.2 produces approximations* $\widetilde{\Phi}(\mathbf{x_i})$ *for* $\mathbf{x_i} \in X$ *such that*

$$|\widetilde{\Phi}(\mathbf{x_i}) - \Phi(\mathbf{x_i})| \leq \epsilon \Phi(\mathbf{x_i}) + \tau \tag{7.4.3}$$

with the probability of at least $1 - \alpha$ *for* $0 < \alpha < 1$*, as the number of samples in the Monte Carlo approximation tends to infinity.*

Proof. We extend the proof in Theorem 7.4.1. For simplicity, we again focus on the $n = 3$ case.

Base case: Given the set of three nodes with the desired failure probability α, the base case MTPotentialBase is easily shown to satisfy Equation 7.4.2 with 100 % probability ($> 1 - \alpha$). Similarly, each Monte Carlo prune satisfies Equation 7.4.2 with probability of $1 - \alpha$ asymptotically.

Inductive case: For a non-prunable set of three nodes $\{\mathbf{P_k}\}_{k=1}^{3}$ for the required failure probability β. Note that MTPotentialCanonical results in a maximum of four (i.e. $2^{3-1} = 4$) sub-calls for a set of non-prunable $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ nodes. For example, suppose $\mathbf{P_1}$ is an internal node, and consider its left child, ${\mathbf{P_1}}^L$. The contribution of $\mathbf{P_2}$ and $\mathbf{P_3}$ on ${\mathbf{P_1}}^L$ can be computed by considering the node combinations: $({\mathbf{P_1}}^L, {\mathbf{P_2}}^L, {\mathbf{P_3}}^L)$, $({\mathbf{P_1}}^L, {\mathbf{P_2}}^L, {\mathbf{P_3}}^R)$, $({\mathbf{P_1}}^L, {\mathbf{P_2}}^R, {\mathbf{P_3}}^L)$, $({\mathbf{P_1}}^L, {\mathbf{P_2}}^R, {\mathbf{P_3}}^R)$, resulting in a maximum of four combinations if $\mathbf{P_1}$, $\mathbf{P_2}$, $\mathbf{P_3}$ satisfy the case 40(a) in Figure 40. Each recursive sub-call is equivalent to a stratum in a stratified sampling, and satisfies the following:

$$\left|\widetilde{\Phi}(\mathbf{x_{i_u}}; {\mathbf{P_2}}^L \times {\mathbf{P_3}}^L) - \Phi(\mathbf{x_{i_u}}; {\mathbf{P_2}}^L \times {\mathbf{P_3}}^L)\right| \leq \frac{\epsilon T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^L \times {\mathbf{P_3}}^L}}{T^{root}} \Phi^l({\mathbf{P_1}}^L) + \frac{\tau T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^L \times {\mathbf{P_3}}^L}}{T^{root}}$$

$$\left|\widetilde{\Phi}(\mathbf{x_{i_u}}; {\mathbf{P_2}}^L \times {\mathbf{P_3}}^R) - \Phi(\mathbf{x_{i_u}}; {\mathbf{P_2}}^L \times {\mathbf{P_3}}^R)\right| \leq \frac{\epsilon T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^L \times {\mathbf{P_3}}^R}}{T^{root}} \Phi^l({\mathbf{P_1}}^L) + \frac{\tau T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^R \times {\mathbf{P_3}}^L}}{T^{root}}$$

$$\left|\widetilde{\Phi}(\mathbf{x_{i_u}}; {\mathbf{P_2}}^R \times {\mathbf{P_3}}^L) - \Phi(\mathbf{x_{i_u}}; {\mathbf{P_2}}^R \times {\mathbf{P_3}}^L)\right| \leq \frac{\epsilon T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^R \times {\mathbf{P_3}}^L}}{T^{root}} \Phi^l({\mathbf{P_1}}^L) + \frac{\tau T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^R \times {\mathbf{P_3}}^R}}{T^{root}}$$

$$\left|\widetilde{\Phi}(\mathbf{x_{i_u}}; {\mathbf{P_2}}^R \times {\mathbf{P_3}}^R) - \Phi(\mathbf{x_{i_u}}; {\mathbf{P_2}}^R \times {\mathbf{P_3}}^R)\right| \leq \frac{\epsilon T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^R \times {\mathbf{P_3}}^R}}{T^{root}} \Phi^l({\mathbf{P_1}}^L) + \frac{\tau T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^R \times {\mathbf{P_3}}^R}}{T^{root}}$$

Collectively, the results from these strata add up to potential estimates that satisfy the error bound with at least $1 - \alpha$ probability for each $\mathbf{x_{i_u}} \in {\mathbf{P_1}}^L$ and the following holds:

$$\left|\widetilde{\Phi}(\mathbf{x_{i_u}}; \mathbf{P_2} \times \mathbf{P_3}) - \Phi(\mathbf{x_{i_u}}; \mathbf{P_2} \times \mathbf{P_3})\right| \leq \frac{\epsilon T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}} \Phi^l(\mathbf{x_{i_u}}) + \frac{\tau T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}}}{T^{root}}$$

where $T_{\mathbf{x_{i_u}} \times \mathbf{P_2} \times \mathbf{P_3}} = T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^L \times {\mathbf{P_3}}^L} + T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^L \times {\mathbf{P_3}}^R} + T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^R \times {\mathbf{P_3}}^L} + T_{\mathbf{x_{i_u}} \times {\mathbf{P_2}}^R \times {\mathbf{P_3}}^R}$. The similar bounds hold for each $x \in {\mathbf{P_1}}^R$, and the same reasoning can be extended to the bounds for $\mathbf{P_2}$ and $\mathbf{P_3}$. Because $\Phi^l(\mathbf{P_1}) = \min\{\Phi^l({\mathbf{P_1}}^L), \Phi^l({\mathbf{P_1}}^R)\}$ throughout the execution of the algorithm, we can extend the argument to the case where $\mathbf{P_1} = \mathbf{P_2} = \mathbf{P_3} = \mathbf{X}$. □

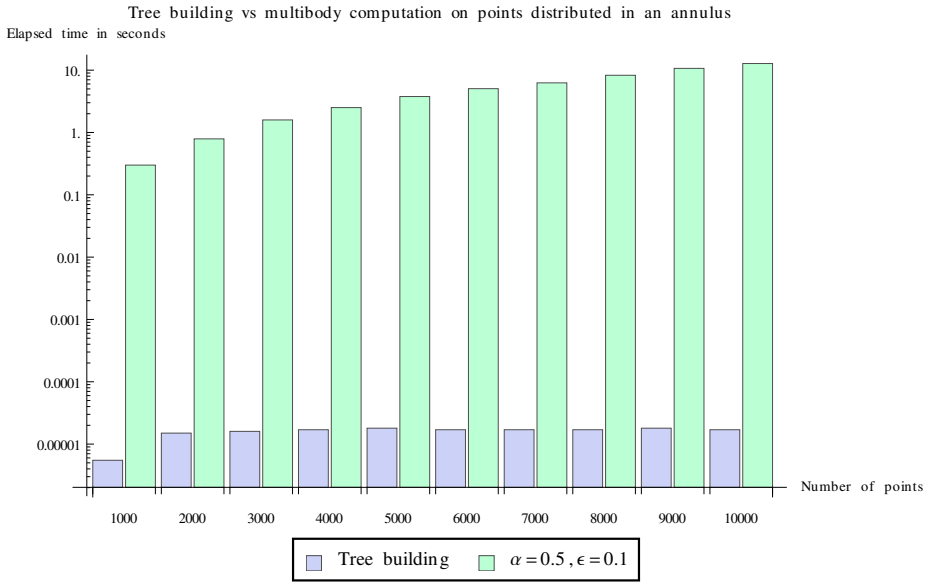


Figure 44: Building the *kd*-tree takes negligible amount of time compared to the time it takes for the actual multibody computation.

7.5 Experiment Results

All of our algorithms were based on an open-source C++ library called MLPACK [24, 53]. The experiments were performed on a desktop with AMD Phenom II X6 1100T Processors utilizing only one core with 8 GB of RAM.

7.5.1 Tree Building

The cost of tree-building is negligible compared to the actual multibody computation. Compared to complex, irregular memory access patterns encountered in the multibody computation (as do most recursive algorithms in general), the tree-building phase requires mostly sequential scanning of contiguous blocks of memory and thus requires shorter amount of time. See Figure 44, where the tree building is compared to the multibody computation with the relative error criterion $\epsilon = 0.1$ and the 50 % probability guarantee ($\alpha = 0.5$). The annulus distribution was chosen deliberately to show that even under the distribution for which the multibody computation is relatively fast (see Section 7.5.2), the tree building requires a tiny fraction of time

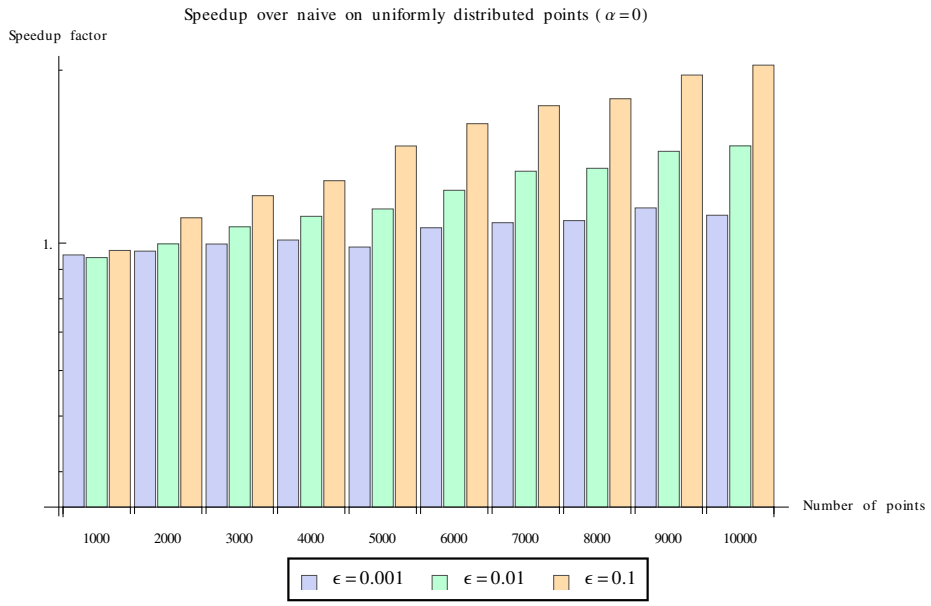


Figure 45: Speedup result on uniformly distributed points using the deterministic algorithm ($\alpha = 0$). The base timings for the naive algorithm on each point set are: 1.91×10^1 seconds, 1.54×10^2 seconds, 5.17×10^2 seconds, 1.23×10^3 seconds, 2.39×10^3 seconds, 4.16×10^3 seconds, 6.64×10^3 seconds, 9.76×10^3 seconds, 1.43×10^4 seconds, and 1.92×10^4 seconds.

compared to the computation time.

7.5.2 Multibody Computation

We demonstrate speedup results of our approximate algorithms guaranteeing the $(1 - \alpha)$ probabilistic ϵ relative error criterion (Definition 2.4.3). For this paper, we focus strictly on the relative error criterion ($\tau = 0$) and test on three relative error parameter values ($\epsilon = 0.001$, $\epsilon = 0.01$, and $\epsilon = 0.1$). We test on three different types of distribution: uniform within the unit hypercube $[0, 1]^3$ (denoted as the "uniform" distribution), the annulus distribution (denoted as the "annulus" distribution) in three dimensions, and uniform within the unit three-dimensional sphere (denoted as the "ball" distribution). These three distributions were also used in [17]).

Deterministic Approximations. Figure 45, Figure 47, and Figure 46 show speedup results against the naive algorithm using only the deterministic approximation (i.e. $\alpha = 0$). On the uniform distribution and the ball distribution, the speedup is almost

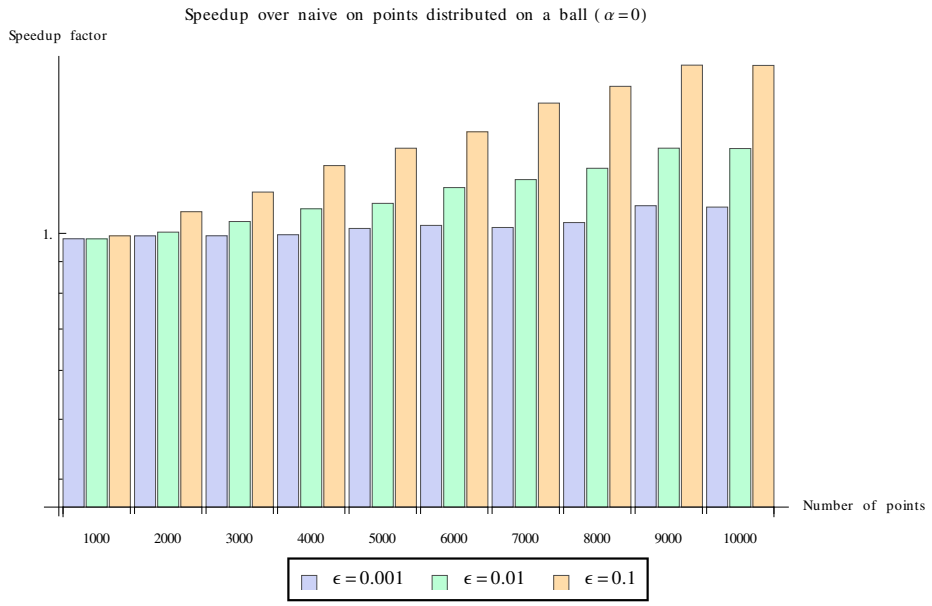


Figure 46: Speedup result on points distributed inside a sphere using the deterministic algorithm ($\alpha = 0$). The base timings for the naive algorithms are listed in Figure 45.

non-existent; the speedup factor is a little bit more than two on the dataset containing $10,000$ points using the lowest parameter setting of $\epsilon = 0.1$. On the annulus distribution, our deterministic algorithm achieves a little bit better speedup against the naive algorithm; a factor of more than 20 times speedup on $10,000$ points is encountered on $\epsilon = 0.1$. A tree-based hierarchical method generally works better for clustered point sets, and this is reflected in our results.

Monte-Carlo Approximations. In this section, we show whether adding indeterminism by sampling can reduce the computation time while guaranteeing a slightly relaxed error criterion (but with a high probability guarantee for each potential sum). We first relax the probability guarantee to be 90% (i.e. $\alpha = 0.1$). Like the results shown using the deterministic algorithm, our Monte Carlo-based algorithm achieves the most speedup on points distributed in an annulus (1000 times speedup on $10,000$ points using $\epsilon = 0.1$).

We also list the percentage of the points actually achieving the ϵ relative error

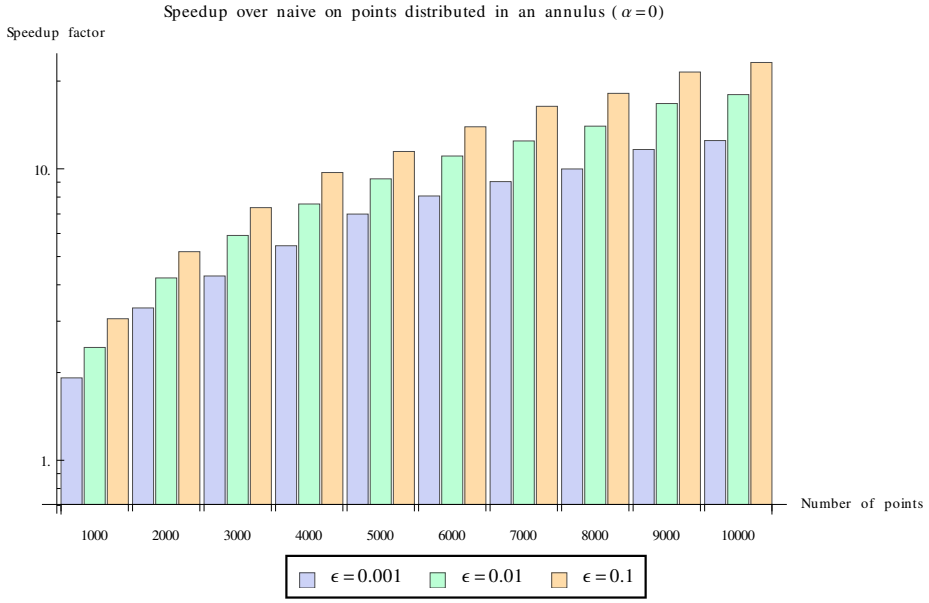


Figure 47: Speedup result on points distributed on an annulus using the deterministic algorithm ($\alpha = 0$). The base timings for the naive algorithms are listed in Figure 45.

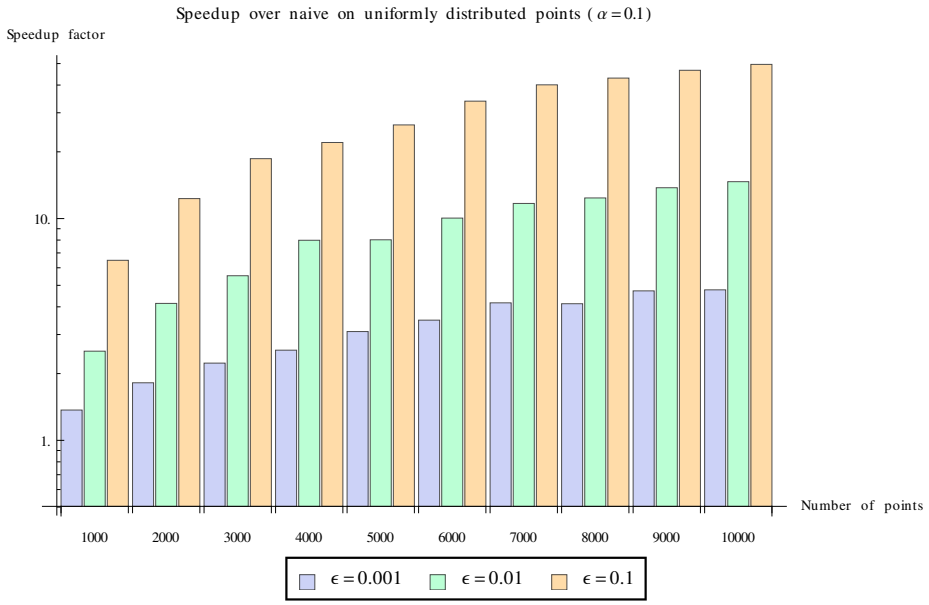


Figure 48: Speedup result on uniformly distributed points using the Monte Carlo-based algorithm ($\alpha = 0.1$). The base timings for the naive algorithms are listed in Figure 45.

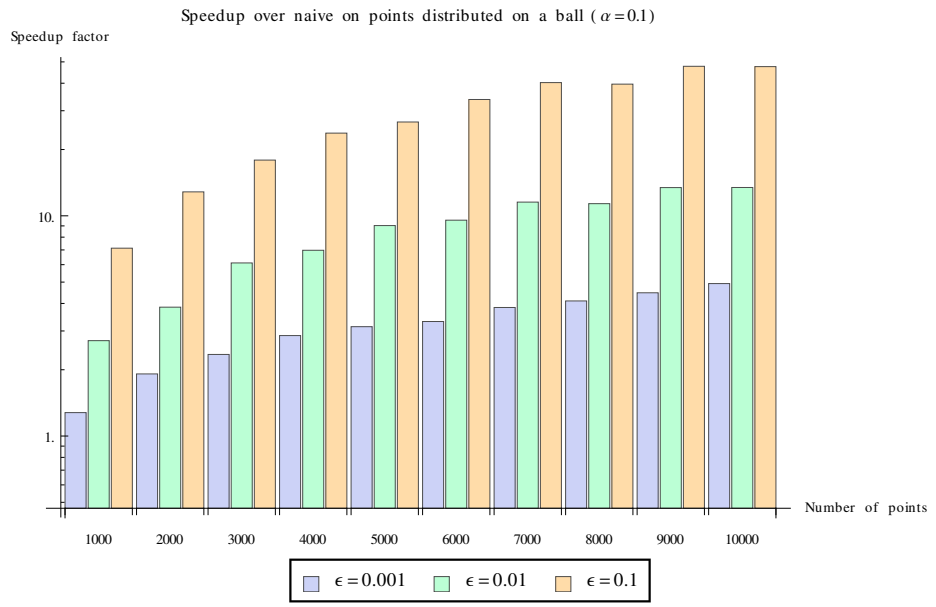


Figure 49: Speedup result on points distributed inside a sphere using the Monte Carlo-based algorithm ($\alpha = 0.1$). The base timings for the naive algorithms are listed in Figure 45.

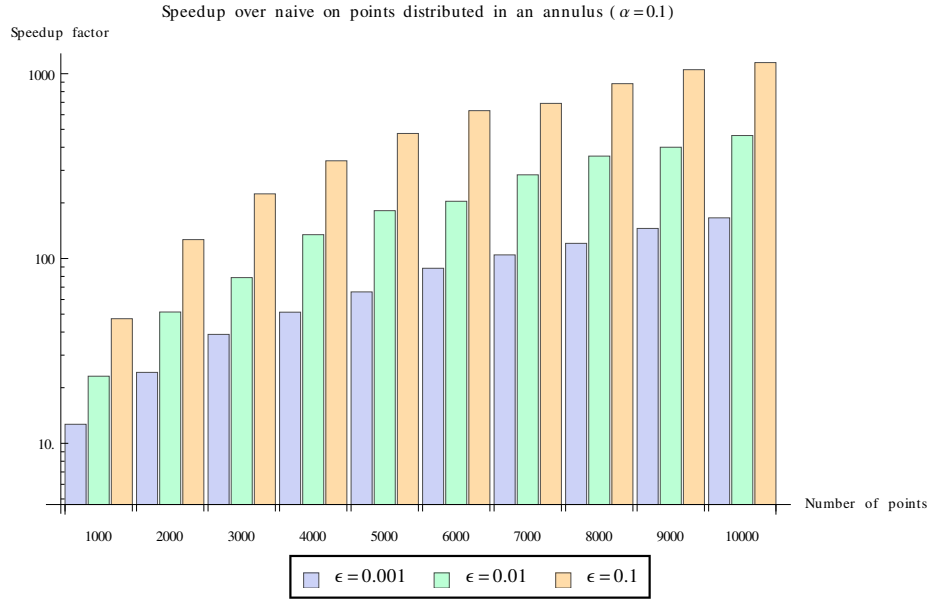


Figure 50: Speedup result on points distributed on an annulus using the Monte Carlo-based algorithm ($\alpha = 0.1$). The base timings for the naive algorithms are listed in Figure 45.

Table 10: The distribution of relative error on the uniform distribution using $\alpha = 0.1$ and $\epsilon = 0.001$.

Number of points	% achieving	Average relative error	Variance	Maximum relative error
1000	98.3%	1.11×10^{-4}	8.89×10^{-7}	2.86×10^{-2}
2000	97.9%	1.28×10^{-4}	7.71×10^{-7}	2.78×10^{-2}
3000	98.6%	1.51×10^{-4}	2.64×10^{-6}	6.47×10^{-2}
4000	98.3%	1.44×10^{-4}	3.37×10^{-6}	1.01×10^{-1}
5000	98.7%	2.65×10^{-4}	1.09×10^{-4}	7.36×10^{-1}
6000	98.3%	1.29×10^{-4}	1.39×10^{-6}	3.62×10^{-2}
7000	98.4%	1.86×10^{-4}	9.29×10^{-6}	1.96×10^{-1}
8000	98.8%	9.89×10^{-5}	1.21×10^{-6}	6.50×10^{-2}
9000	98.8%	9.94×10^{-5}	1.39×10^{-6}	6.69×10^{-2}
10000	98.9%	1.02×10^{-4}	1.95×10^{-6}	1.06×10^{-1}

bound along with the mean and the variance in Table 10, Table 11, and Table 12. The relative error level of 0.001 and the probability guarantee of 90% was used. Under all three distributions, the percentage of points whose potential sum achieved the desired relative error of 0.001 was well above 90%. We list the average relative error, the variance, and the maximum relative error. Note that the maximum relative error can exceed 100% if the true potential sum and its approximation have opposite signs. For a particle with a small potential sum, we have observed that this is indeed the case due to numerical inaccuracies accumulated during the summation.

7.6 Conclusion

In this chapter, we have introduced the framework for extending the pairwise series expansion to potentials that involve more than two points. Through this process, we have formally defined the expansions needed for approximating the multibody potentials in a hierarchical fashion as done in traditional FMM algorithms, and have derived algorithms for guaranteeing (1) absolute error bound (2) relative error bound (3) probabilistic absolute/relative error on each particle potential sum and proved the correctness of our algorithms formally. Parallelization is left as a future work [118].

Table 11: The distribution of relative error on the ball distribution using $\alpha = 0.1$ and $\epsilon = 0.001$.

Number of points	% achieving	Average relative error	Variance	Maximum relative error
1000	98.6%	8.21×10^{-5}	1.17×10^{-7}	7.22×10^{-3}
2000	98.7%	1.35×10^{-4}	1.26×10^{-6}	2.78×10^{-2}
3000	98.7%	1.11×10^{-4}	7.58×10^{-7}	3.23×10^{-2}
4000	97.0%	1.36×10^{-3}	1.21×10^{-3}	1.81×10^{0}
5000	98.2%	1.19×10^{-4}	1.18×10^{-6}	4.85×10^{-2}
6000	98.9%	1.20×10^{-4}	3.70×10^{-6}	1.27×10^{-1}
7000	98.8%	1.22×10^{-4}	3.32×10^{-6}	1.11×10^{-1}
8000	98.5%	1.31×10^{-4}	3.67×10^{-6}	1.12×10^{-1}
9000	97.9%	6.24×10^{-4}	3.89×10^{-4}	1.14×10^{0}
10000	97.6%	5.09×10^{-4}	2.40×10^{-4}	1.28×10^{0}

Table 12: The distribution of relative error on the annulus distribution using $\alpha = 0.1$ and $\epsilon = 0.001$.

Number of points	% achieving	Average relative error	Variance	Maximum relative error
1000	98.4%	9.33×10^{-5}	3.42×10^{-7}	1.38×10^{-2}
2000	97.2%	9.21×10^{-4}	2.69×10^{-4}	5.15×10^{-1}
3000	98.7%	8.52×10^{-5}	1.16×10^{-6}	5.09×10^{-2}
4000	91.8%	2.53×10^{-2}	6.10×10^{-1}	4.80×10^{1}
5000	96.9%	1.28×10^{-3}	1.09×10^{-3}	1.27×10^{0}
6000	92.8%	6.28×10^{-3}	1.38×10^{-2}	6.43×10^{0}
7000	95.2%	2.13×10^{-3}	1.36×10^{-3}	6.66×10^{-4}
8000	91.2%	1.45×10^{-2}	3.77×10^{-1}	5.36×10^{1}
9000	94.6%	5.17×10^{-3}	6.56×10^{-3}	3.94×10^{0}
10000	91.6%	2.72×10^{-2}	8.06×10^{-1}	8.29×10^{1}

CHAPTER VIII

BEYOND SERIAL IMPLEMENTATIONS: DISTRIBUTED FAST SUMMATIONS

8.1 Issues in Parallelization

In this chapter, we propose a parallel framework for kernel summations extending the serial approaches described in Chapter 3, Chapter 4, and Chapter 5[1]. The framework provides a general approach for accelerating the computation of many popular machine learning methods. The motivation is similar to that of [119] and [98]. In [119], a general framework was developed to support various types of scientific simulations. In the PEGASUS framework [98], several graph mining operations (PageRank, Random Walk with Restart (RWR), diameter estimation, and connected components) was parallelized via an implementation of *Generalized Iterated Matrix-Vector multiplication* (GIM-V) on HADOOP platform [21]. This chapter is based on parallelization of the previously successful *generalized N-body framework* [78, 128] which is similar to the well-known spatial join algorithms [64, 26] and is an extension of the parallelization work in [25]. We again start by defining the computational task to be tackled.

Problem: Suppose we are given the set of query points $\mathbf{Q}$ and the set of reference points $\mathbf{R}$, and each of these sets are equi-distributed across a network of nodes. Given a pairwise kernel function k, the relative error level $\epsilon > 0$, and the desired kernel sum $\Phi(\mathbf{q}; \mathbf{R}) = \sum_{\mathbf{r} \in \mathbf{R}} k(\mathbf{q}, \mathbf{r})$ for each $\mathbf{q} \in \mathbf{Q}$,
Task: Compute an approximation $\widetilde{\Phi}(\mathbf{q}; \mathbf{R})$ for each $\mathbf{q} \in \mathbf{Q}$ such that $\left| \widetilde{\Phi}(\mathbf{q}; \mathbf{R}) - \Phi(\mathbf{q}; \mathbf{R}) \right| \leq \epsilon \Phi(\mathbf{q}; \mathbf{R})$ as fast as possible.

[1] The extension to the multibody case is under progress in another submission.

Table 13: Methods that can be sped up using our framework. Although the parts marked with ✕ can be sped up in some cases by sparsifying the kernel matrix and applying Krylov-subspace methods, computed results are usually numerically unstable. An alternative approach based on distributed averaging and random feature extraction will be introduced in Chapter 9.

Method	$k(\cdot,\cdot)$	Train/Batch test
KDE [145]/NWR [138]	PDFs	✔ / ✔
KSVM [165]/GPR [153]	PD kernels	✕ / ✔
KPCA [164]	CPD kernels	✕ / ✔

We now start by defining the necessary terminologies.

- An *MPI communicator* connects a set of MPI processes, each of which is given a unique identifier called an *MPI rank*, in an ordered *topology*
- Commonly used topologies include: the ring topology, the star topology, and the hypercube topology. We denote C_{world} as the MPI communicator over all MPI processes, and $\mathbf{D_P}$ the portion of the data $\mathbf{D}$ owned by the P-th process.

In this chapter, we assume that: 1) the nodes are connected using a hypercube topology since it is the most commonly used one; 2) there are p_{thread} threads associated with each MPI process; 3) the number of MPI processes p is a power of two[2], though our approach can be easily extended for arbitrary positive integers p; 4) the query set equals the reference set ($\mathbf{Q} = \mathbf{R}$, and we denote $\mathbf{D}$ as the common dataset and $N = |\mathbf{D}|$ the size of the dataset), and $\mathbf{D}$ is equi-distributed across all MPI processes. Particularly the monochromatic case of $\mathbf{Q} = \mathbf{R}$ occurs often in cross-validating for optimal parameters in many non-parametric methods.

Hierarchical N-body methods present an interesting challenge in parallelization: 1) both data distribution and work distribution are highly non-uniform across MPI processes; 2) often involves long-range communication due to the kernel function

[2]We regret the potential overloading of notation here. p has been used in Chapter 3 and Chapter 4 as truncation orders.

Table 14: Examples of approximation schemes that can be utilized in our framework.

Approximation	Type	Basis functions	Applicability
Series expansion [114, 112]	Deterministic	Taylor basis	General
Reduced set [165]	Deterministic	Pseudo-particles	Low-rank PD/CPD kernels
Monte Carlo [95, 113]	Probabilistic	None	General smooth kernels
Random feature extraction [150]	Probabilistic	Fourier basis	Low-rank PD/CPD kernels

Table 15: Examples of multi-dimensional binary trees that can be utilized in our framework. If RULE($\mathbf{x}$) returns true, then x is assigned to the left child (as defined in [54]).

Tree type	Bound type	RULE($\mathbf{x}$)
kd-trees [15]	hyper-rectangle $\{\mathbf{b}^{min}[d], \mathbf{b}^{\max}[d]\}_{d=1}^{D}$	$\mathbf{x}[i] \leq \mathbf{s}[i]$ for $1 \leq i \leq D$, $\mathbf{b}^{min}[i] \leq \mathbf{s}[i] \leq \mathbf{b}^{max}[i]$
metric trees [140]	hyper-sphere $B(\mathbf{c}, r)$, $\mathbf{c} \in \mathbb{R}^D, r > 0$	$\|\|\mathbf{x}-\mathbf{p}_{left}\|\| < \|\|\mathbf{x}-\mathbf{p}_{right}\|\|$ for $\mathbf{p}_{left}, \mathbf{p}_{right} \in \mathbb{R}^D$
vp-trees [202]	$B(\mathbf{c}, r_1) \cap B(\mathbf{c}, r_2)$ for $0 \leq r_1 < r_2$	$\|\|\mathbf{x} - \mathbf{p}\|\| < t$ for $t > 0, p \in \mathbb{R}^D$
RP-trees [54]	Hyperplane $\mathbf{a}^T\mathbf{x} = b$	$\mathbf{x}^T\mathbf{v} \leq$ MEDIAN($\mathbf{z}^T\mathbf{v} : \mathbf{z} \in \mathbf{S}$)

$k(\cdot, \cdot)$. In the worst case, every process will need almost every piece of data owned by the other processes. Here we discuss the three main important issues in a scalable distributed hierarchical N-body code:

Parallel Tree Building: [110] proposed a novel distributed octree construction algorithm and a new reduction algorithm for evaluation to scale up to over 65K cores. [3] describes a parallel *kd*-tree construction on a distributed memory setting, while [39] works on a shared-memory setting. [120] discuss building spill-trees, a variant of metric trees that permit overlapping of data between two branches, using the MapReduce framework.

Load Balancing: Most common static load balancing algorithms include: 1) the costzone [173] which partitions a pre-built query tree and assigns each query particle to a zone. A common approach employs a graph partitioner [52]; 2) the ORB (orthogonal recursive bisection) which directly partitions each dimension of the space

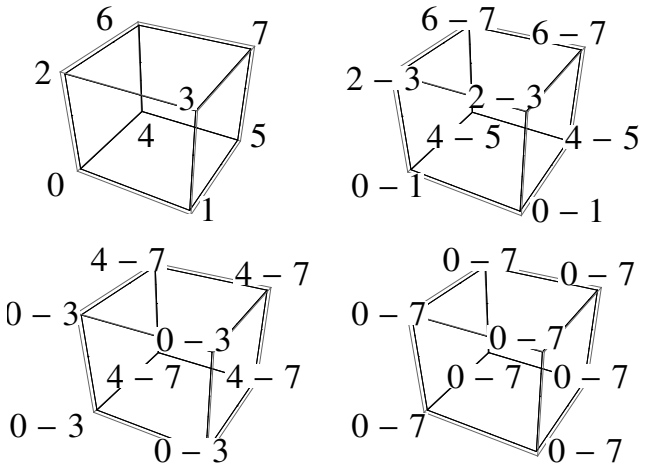


Figure 51: Recursive doubling on the hypercube topology. Initially, each node begins with its own message (top left). The exchanges proceed in: the top right, the bottom left, then bottom right in order. Note that the amount of data exchanged in each stage doubles.

containing the query points in a cyclic fashion. Dynamic load balancing [123] strategies adjust the imbalance between the work loads during the computation.

Interprocess Communication: The local essential trees approach [160] (which involves few large-grained communication) is a sender-initiated communication approach. Using the ORB, each process sends out essential data that may be needed by the other processes using the recursive doubling scheme (see Figure 51). An alternative approach has the receiver initiate communication; this approach involves many fine-grained communication and is preferable if interprocess communication overheads are small. For more details, see [172].

8.2 Distributed Multidimensional Tree

Our approach for building a *general-dimension* distributed tree closely follows [3]. Following the ORB (orthogonal recursive bisection) in [160], we define the *global tree*, which is a hierarchical decomposition of the data points on the process level. The *local tree* of each process is built on its own local data D_P.

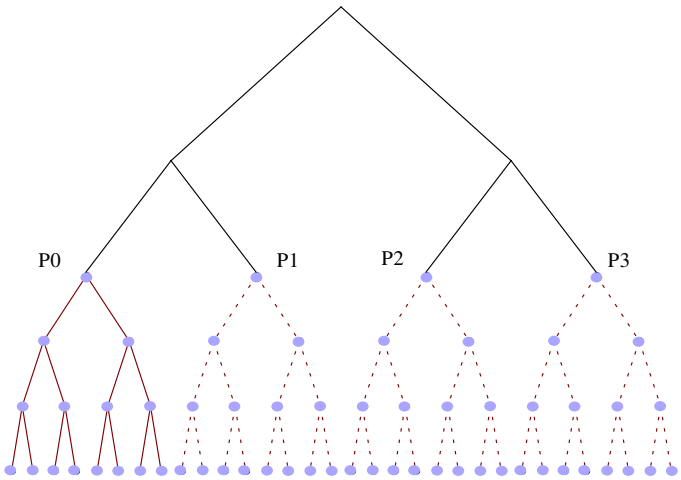


Figure 52: Each process owns the global tree of processes (the top part) and its own local tree (the bottom part).

Building the Distributed Tree. Initially, all MPI processes in a common MPI communicator agree on a rule for partitioning each of its data into two parts (see Algorithm 8.2.1). The MPI communicator is then split in two depending on the MPI process rank. This process is recursively repeated until there are $\log p$ levels in the global tree. Shared-memory parallelism can be utilized in the (independent) reduction step in each MPI process in generating the split rule (see Algorithm 8.3.2). Using C++ meta-programming, we can auto-generate any binary tree utilizing an associative reduction operator for constructing bounding primitives; one just needs to provide a splitting rule (see Table 15). Generalizing to multidimensional trees with an arbitrary number of child nodes (such as cover-trees [20]) is left as a future work.

Building the Local Tree. Here we closely follow the approach in [39]. The first few levels of the tree are built in a breadth-first manner with the assigned number of OpenMP threads proportional to the number of points participating in a reduction to form the bounding primitive (see Figure 54). The number of participating OpenMP threads per task halves as we descend each level. Each independent task with only one assigned OpenMP thread proceeds with the construction in a depth-first manner.

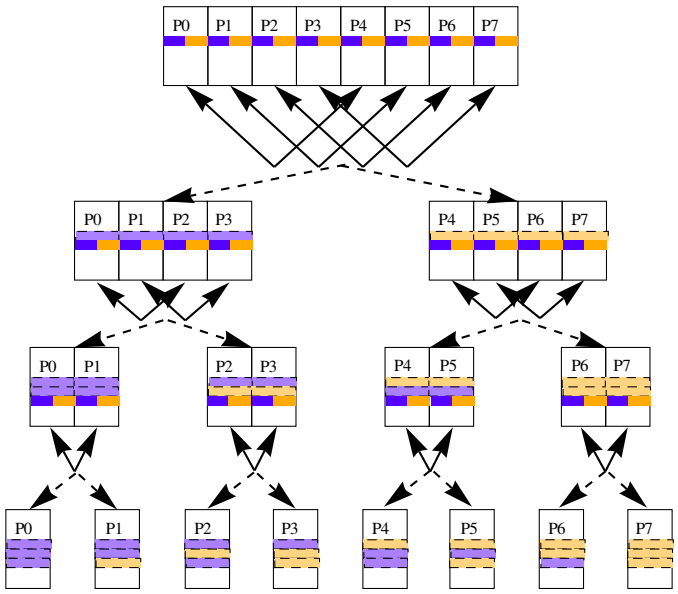


Figure 53: Distributed memory parallelism in building the global tree (the first $\log p$ levels of the entire tree). Each solid arrow indicates a data exchange between two given processes. After exchanges on each level, the MPI communicator is split (shown as a dashed arrow) and the construction works in parallel subsequently.

We utilized the nested loop parallelization feature in OpenMP for this part. Recently, we have transitioned to use Intel Thread Building Block for its simplicity in recursive task-based parallelism.

Overall Runtime Complexity. All-reduce operation on the hypercube topology takes $\mathcal{O}\left(t_s \log p + t_w m(p-1)\right)$ where t_s, t_w, and m are the latency constant, the bandwidth constant, and the message size respectively. Assume that each process starts with the same number of points $\frac{N}{p}$ and each split on a global/local level results in equi-distribution of points and only distributed memory parallelism is used (i.e. $p_{thread} = 1$). Let m_{bound} be the message size of the bounding primitive divided by D. The overall runtime for each MPI process is:

- The reduction cost and the split cost at each level $0 \leq i < \log p$: $\mathcal{O}\left(\frac{2N(D+t_w)}{p}\right)$
- The all-reduce cost on each level $0 \leq i < \log p$: $\mathcal{O}(t_w D m_{bound}\left(\frac{p}{2^i} - 1\right))$
- The total latency cost at each level $0 \leq i < \log p$: $\mathcal{O}\left(t_s\left(\log \frac{p}{2^i} + 1\right)\right)$.

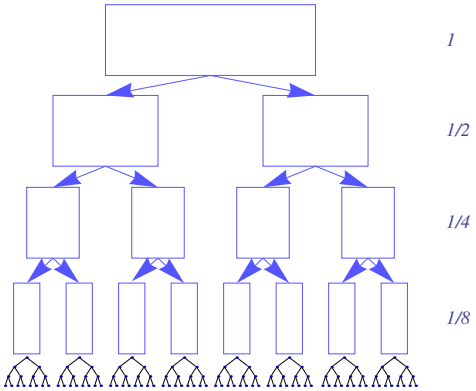


Figure 54: Shared-memory parallelism in building the local tree for each MPI process. The first top levels are built in a breadth-first manner with the number of threads proportional to the amount of performed reduction. Any task with one assigned thread proceeds in a depth-first manner.

- The base case at the level $\log p$ (the depth-first build of local tree): $\mathcal{O}\left(\frac{DN}{p}\log\left(\frac{N}{p}\right)\right)$

Therefore, the overall complexity is: $\mathcal{O}\left(\frac{DN}{p}\log\left(\frac{N}{p}\right)\right)+\mathcal{O}\left(Dt_w m_{bound}\left(2p-\log 4p\right)\right)+\mathcal{O}\left(\frac{2N(D+t_w)}{p}\log p\right)+\mathcal{O}\left(\frac{t_s}{2}\log p\left(\log p+3\right)\right)$. This implies that the growth of the number of data points must be $N\log N \sim \mathcal{O}(p^2)$ to achieve the same level of parallel efficiency. Note that the last terms have zero contribution if $p=1$.

8.3 Overall Algorithm

Algorithm 8.3.1 shows the overall algorithm. Initially, each MPI process initializes its distributed task queue by dividing its own local query subtree into a set of T query grain subtrees where $T > p_{thread}$ is more than the number of threads p_{thread} running on each MPI process; initially each of these trees has no tasks. The tree walker object maintains a stack of pairs of $\mathbf{Q}^{sub}$ and $\mathbf{R_P}^{sub}$ that must be considered. It is first initialized with the following tuple: the root node of $\mathbf{Q}$, the root node of the local reference tree $\mathbf{R_P}$, and the probability guarantee α; the relative error tolerance/absolute error tolerance are global constants ϵ and τ respectively. Threads not involved with the tree walk or exchanging data can dequeue tasks from the local

Algorithm 8.2.1 BUILDDISTTREE($C_{world}, \mathbf{D_P}$): (**MPI**)

```
C ← C_world
while C.size() > 1 do
  rule ← CHOOSESPLITRULE(C, D_P)
  for each P-th MPI process in C in parallel do
    Divide D_P = L_P ∪ R_P using rule.
    if P < |C|/2 then
      P_comp ← P + |C|/2,            SEND(P_comp, R_P)
      L_comp ← RECEIVE(P_comp), D_p ← L_p ∪ L_comp
    else
      P_comp ← P − |C|/2, R_comp ← RECEIVE(P_comp)
      SEND(P_comp, L_P), D_p ← R_p ∪ R_comp
    C ← SPLITCOMM(P >= |C|/2)
BUILDLOCALTREE(D_P)
```

task queue.

8.3.1 Walking the Trees

Each MPI process takes the root node of the global query tree (the left tree) and the root node of its local reference tree (the right tree) and performs a dual-tree recursion (see Algorithm 8.3.3). For simplicity, we show the case where the reference side is descended first then the query side. Any of the running threads can walk by dequeuing from the stack of frontier nodes, generate local tasks, and queue up reference subtrees to send to other processes. The expansion can be prioritized using the HEURISTIC function that takes a pair of query/reference nodes. It would be possible to extend the walking procedure to include fancier expansion patterns described in [156].

8.3.2 Message Passing

Inspired by the local essential trees approach, we develop a message passing system utilizing the *recursive doubling scheme*. We assume that the master thread is the only thread that may initiate MPI calls. The key differences from the vanilla local essential tree approach are two-fold: 1) our framework can support computations that have dynamic work requirement, unlike FMM; 2) our framework does

Algorithm 8.3.1 Overall algorithm.

Each MPI process initializes its distributed task queue with a set of query grain subtrees and the tree walker with $(\mathbf{Q}, \mathbf{R_P})$.
OpenMP parallel region start (threads spawned)
while there are remaining tasks globally **do**
  if I am the master thread **then**
    Route messages via recursive doubling.
  if If the task queue is nearly empty **then**
    WALK() (Algorithm 8.3.3,Figure56).
  Choose a query subtree and lock it. Dequeue a set of task from it and call the serial algorithm (Algorithm 3.1.10 on each $(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$ pair. For each completed task, queue up completed work quantity.

  Unlock the query subtree. If the checked out query subtree is imported from another process and has no more tasks, queue up a flush request to write back the query subtree.
OpenMP parallel region end (threads synchronized)

Algorithm 8.3.2 CHOOSESPLITRULE$(C, \mathbf{D_P})$: (**OpenMP/IntelTBB**)

$\mathbf{b}_{local} \leftarrow$ an empty bound
for each data point $\mathbf{r} \in \mathbf{D_P}$ in parallel **do**
  Expand $\mathbf{b}_{local}$ to include $\mathbf{r}$.
$\mathbf{b}_{common} \leftarrow$ COMBINE$(C, \mathbf{b}_{local})$
return RULE$(\mathbf{x})$ using $\mathbf{b}_{common}$.

not require each MPI process to accommodate all of the non-local data in its essential tree. Algorithm 8.3.4 shows the message passing routine called by the master threshold on each MPI process. Any message from a pair of processes in a hypercube topology needs at most $\log p$ rounds of routing. At each stage i, the process P with binary representation $P = (b_{\log(p)-1}, \cdots, b_{i+1}, 0, b_{i-1}, \cdots, b_0)_2$ sends messages to process $P_{neighbor} = (b_{\log(p)-1}, \cdots, b_{i+1}, 1, b_{i-1}, \cdots, b_0)_2$ (and vice versa). Here are the types of messages exchanged between a pair of processes:

1. Reference subtrees: each MPI process sends out a reference subtree with the tag $(\mathbf{R}^{sub}, \{\mathbf{Q}^{sub}\})$ where $\{\mathbf{Q}^{sub}\}$ is the list of remote query subtrees that needs $\mathbf{R}^{sub}$.

2. Work-complete message: whenever each thread finishes computing a task $(\mathbf{Q}^{sub}, \mathbf{R}^{sub})$,

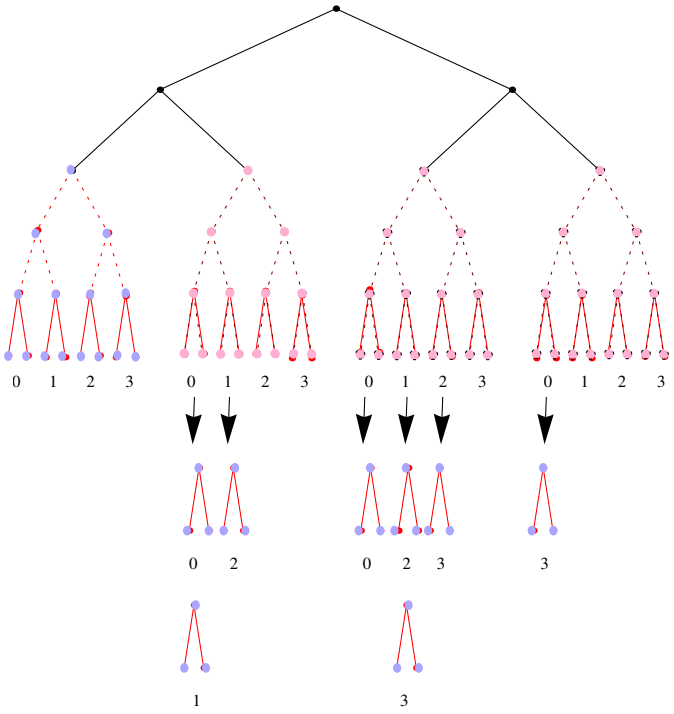


Figure 55: The global query tree is divided into a set of query subtrees each of which can queue up a set of reference subset to compute (shown vertically below each query subtree). The kernel summations for each query subtree can proceed in parallel.

it queues up a pair of completed work quantity and the list of all MPI ranks excluding the self. The form of the message is: $(|\mathbf{Q}^{sub}||\mathbf{R}^{sub}|, \{0, \cdots, P-1, P+1, p-1\}))$.

3. Extra tasks: one of the paired MPI processes can donate some of its tasks to the other (Section 8.3.3). This has a form of $(\mathbf{Q}^{sub}, \{\mathbf{R}^{sub}\})$ where $\{\mathbf{R}^{sub}\}$ is a list of reference subsets that must be computed for $\mathbf{Q}^{sub}$.

4. Imported query subtree flushes: during load balancing, query subtrees with several reference tasks may be imported from another process. These must be synchronized with the original query subtree on its originating process before tasks associated with it are dequeued.

5. The current load: the load is defined as the sum of $|\mathbf{Q}^{sub}\mathbf{R}^{sub}|$ associated with all

Algorithm 8.3.3 WALK()

```
while there is a MPI process asking for work do
  (Q^sub, R^sub) ← POP()
  if CANSUMMARIZE(Q^sub, R^sub) then
    SUMMARIZE(Q^sub, R^sub), Queue the work-complete message
    (|Q^sub||R^sub|, {1, ··· , P − 1, P + 1, ··· , p}).
  else
    if Q^sub is a root node of a query grain subtree then
      if R^sub is a leaf node then
        if Q^sub belongs to the self, then
          Add (Q^sub, R^sub) to the task list of Q^sub.
        else
          Add the MPI rank of Q^sub to a list of R^sub's destinations.
      else
        (R_1, R_2) ← HEURISTIC(Q^sub, R^sub,L, R^sub,R)
        PUSH (Q^sub, R_2), PUSH (Q^sub, R_1)
    else
      if R^sub is a leaf node then
        PUSH(Q^sub,L, R^sub, ), PUSH(Q^sub,R, R^sub)
      else
        (R_L,1, R_L,2) ← HEURISTIC(Q^sub,L, R^sub,L, R^sub,R)
        (R_R,1, R_R,2) ← HEURISTIC(Q^sub,R, R^sub,L, R^sub,R)
        PUSH (Q^sub,R, R_R,2) , PUSH (Q^sub,L, R_L,2)
        PUSH (Q^sub,R, R_R,1) , PUSH (Q^sub,L, R_L,1)
```

query subtrees (both native and imported) on a given process.

Distributed Termination Detection. We follow a similar idea discussed in Section 14.7.4 of [143], Initially, all MPI processes collectively have to complete $|\mathbf{Q}||\mathbf{R}|$ amount of work. Each thread dequeues a work and completes a portion of its assigned local work (see Figure 55); the completed work quantity is then broadcast using the recursive doubling message passing to all the other processes. The completed and uncompleted work is conserved at any given point of time. When every process thinks all of $|\mathbf{Q}||\mathbf{R}|$ work have been completed and it has sent out all of its queued up work-complete messages, it can safely terminate.

8.3.3 Load Balancing

Our framework employs both static load balancing and dynamic load balancing.

Static Load Balancing. Each MPI task is initially in charge of computing the kernel sums for all of its grain query subtrees. This approach is similar to the ORB approach where the distributed tree determines the task distribution.

Dynamic Load Balancing. It is likely that the initial query subtree assignments will cause imbalance among processes. During the computation, we allow each query task to migrate from the current P-th process to a neighboring $P_{neighbor}$-th process. We use a very simple scheme in which two processes that are paired up during each stage of the repeated recursive doubling stages attempt to load balance. Each process keeps sending out a snapshot of its computation load in the recursive doubling scheme, and maintains a table of estimated remaining amount of computation on the other processes. Therefore, load estimates could be outdated by the time a given process considers transferring extra tasks. Therefore, we employ a simple heuristic of initiating the load balance for a pair of imbalanced processes: if the estimated load on the process $P_{neighbor}$ is below $0 < \beta_{threshold} < 1$ of the current load on the process P, transfer $0.5(1 - \beta_{threshold})$ amount of tasks from P to $P_{neighbor}$.

8.4 Experimental Results

We developed our code base in C++ called MLPACK [24] and utilized open-source libraries such as Boost library [107], Armadillo linear algebra library [163], and the GNU Scientific Library [49]. We have tested on the Hopper cluster at NERSC. Each node on the Hopper cluster has 24 cores, and we used the recommended setting of 6 OpenMP threads/node ($p_{thread} = 6$) and a maximum 4 MPI tasks/node and compiled using GNU C++ compiler version 4.6.1 under the $-O3$ optimization flag. The configuration details are available at [1].

We chose to evaluate the scalability of our framework in the context of computing kernel density estimates [145]. We used the Epanechnikov kernel $k(q, r) = I\left(1 - \frac{||q-r||^2}{h^2}\right)$ since it is the most asymptotically optimal kernel. For the first

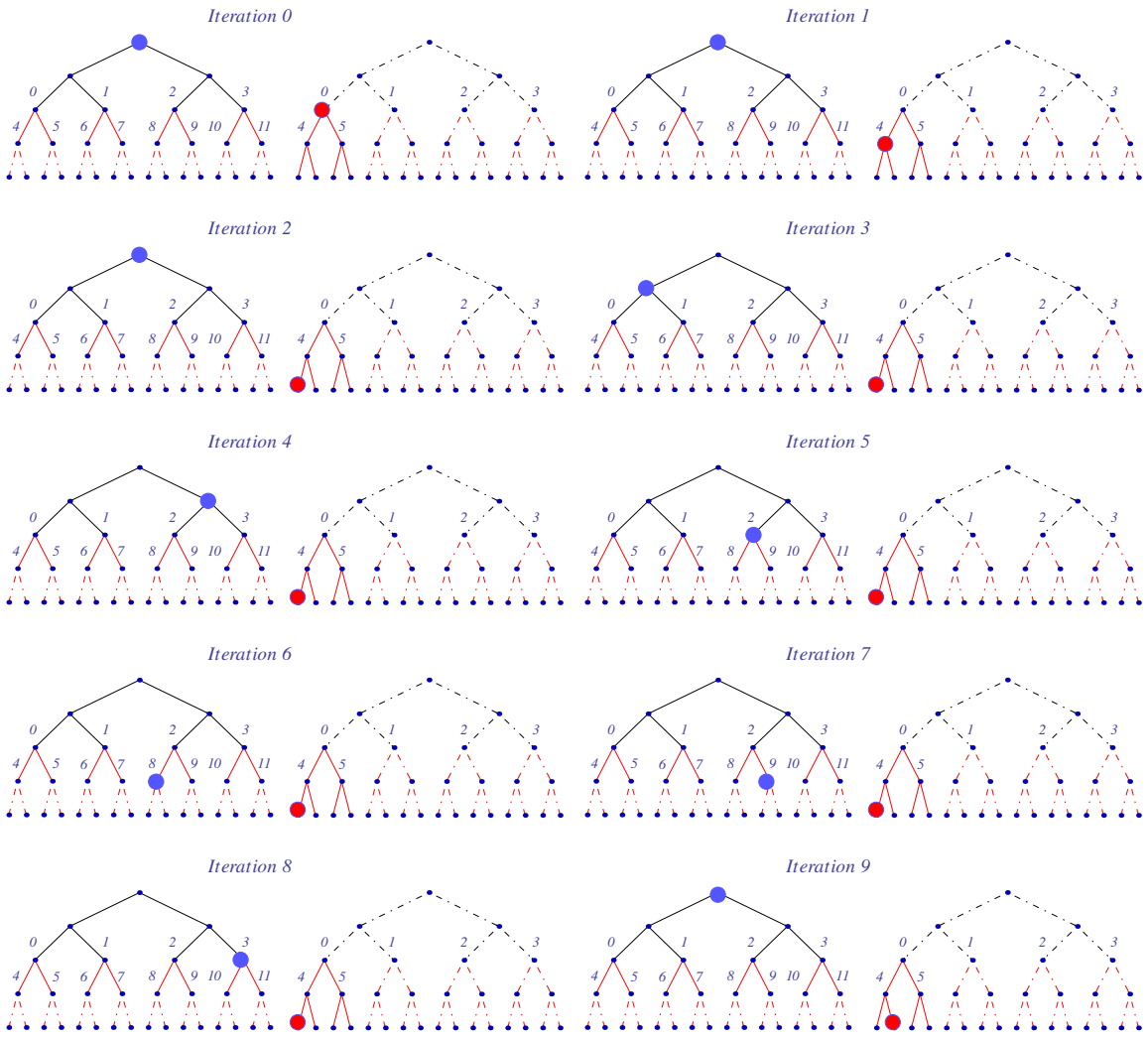


Figure 56: Illustration of the tree walk performed by the 0-th MPI process in a group of 4 MPI processes. Iteration 0: starting with the global query tree root and the root node of the local reference tree owned by the 0-th MPI process; Iteration 1-2: descend the reference side before expanding the query side; Iteration 3: the reference subtree 12 is pruned for the 0-th and 1st MPI processes; Iteration 6-7: the reference subtree 12 is hashed to the list of subtrees to be considered for the query subtrees 8 and 9 (owned by the 2nd MPI process); Iteration 8: the reference subtree 12 is pruned for the 3rd MPI process. Iteration 9: the reference subtree 13 is considered subsequently after the reference subtree 12. At this point, the hashed reference subtree list includes (12, {8, 9}).

Algorithm 8.3.4 RouteMessage

$P_{neighbor} \leftarrow P$ **XOR** *stage*.
Asynchronously send to $P_{neighbor}$:

1. A set of query subtree flushes
2. A set of query subtrees with tasks
3. The work-complete messages
4. The recently received load estimates of other processes.

From $P_{neighbor}$, receive:

1. A set of query subtree flushes from $P_{neighbor}$. Synchronize those that belong to P.
2. Query subtrees with tasks from $P_{neighbor}$ and have the local task queue import them.
3. Load estimates of other processes from $P_{neighbor}$.
4. Work complete messages from $P_{neighbor}$ and update the global work count.

Wait until all sends are complete.
$stage \leftarrow (stage + 1)$ **mod** $\log p$

part of our experiments, we considered uniformly distributed data points in the 10-dimensional hypercube $[0, 1]^{10}$ since non-parametric methods such as KDE and NWR require an exorbitant number of samples in the uniform distribution case. Applying non-parametric methods for higher dimensional datasets requires exploiting correlations between dimensions [142]. For the second part, we measured the strong scalability of our implementation on the SDSS dataset. All timings are maximum ones across all processes.

8.4.1 Scalability of Distributed Tree Building

We have compared the strong scalability of building two main tree structures: *kd*-trees and metric-trees on an uniformly distributed 10-dimensional dataset containing 20,029,440 points (Figure 57). In all cases, building a metric-tree is more expensive than building a *kd*-tree; a reduction operation in Algorithm 8.3.2 for metric-trees

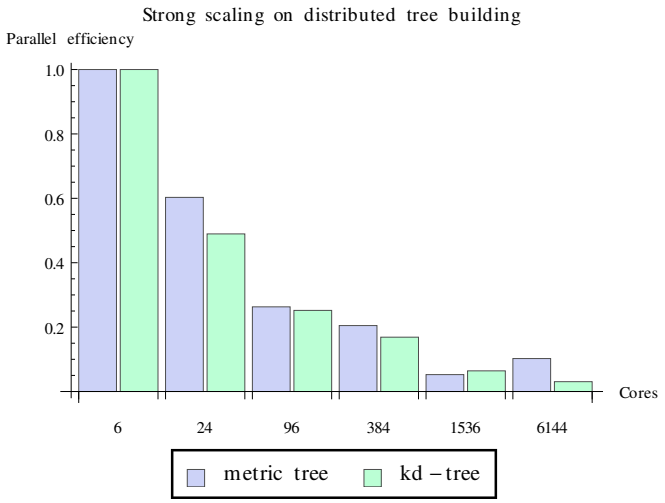


Figure 57: Strong scaling result for distributed tree building on an uniform point distribution in the 10-dimensional unit hypercube $[0, 1]^{10}$. The dataset has 20,029,440 points. The base timings for 6 cores are 105 seconds and 52.9 seconds for metric-tree and kd-tree respectively. The raw timings for all pairs are (in seconds): (104.86, 52.93), (43.48, 27.03), (24.92, 13.13), (8, 4.9), (7.8, 3.22), (6.5, 1.69).

involves distance computations whereas the reduction operator for *kd*-trees is the computation of minimum/maximum. For the weak-scaling result (shown in Figure 58), we added 166,912 ten-dimensional data points per core up to 1,025,507,328 points. Our analysis in Section 8.2 has shown that the exact distributed tree building algorithm require the growth of the data points to be $N \log N \sim O(p^2)$, and this is reflected in our experimental results.

However, readers should note that: 1) the depth of the trees built in our setting is much deeper than the ones in other papers [110]. Each leaf in our tree contains 40 points; 2) the tree building is empirically fast. On 6,144 cores, we were able to build a *kd*-tree on over one billion 10-dimensional data points under 30 seconds; 3) the one-time cost of building the distributed tree can be amortized over many queries.

[120] took a simple map-reduce approach in building a multidimensional binary tree (hybrid spill-trees specifically). We conjecture that this approach may be faster to build but result in slower query times due to generating suboptimal partitions. Future experiments will reveal its strengths and the weaknesses.

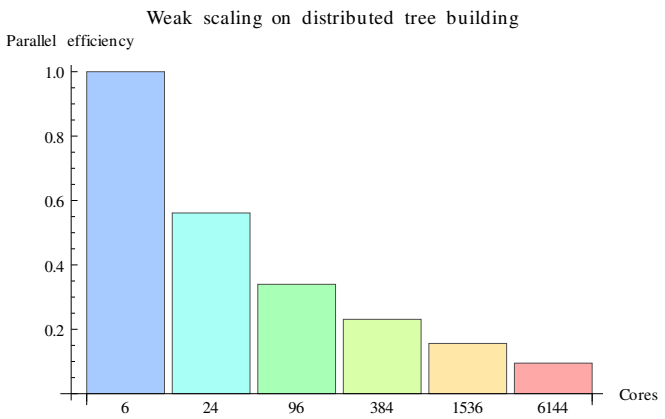


Figure 58: Weak scaling result for distributed *kd*-tree building on an uniform point distribution in 10 dimensions. We used 166,912 points / core. The base timing for 6 cores is 2.81 seconds.

8.4.2 Scalability of Kernel Summation

In this experiment, we measure the scalability of the overall kernel summation. Our algorithm has three main parts: building the distributed tree (Algorithm 8.2.1), walking the tree to generate the tasks (Algorithm 8.3.3,Figure 56), and performing reductions on the generated tasks (Figure 55). The kernel summation algorithm tested here employs only the deterministic approximations [114, 112]. We used $\epsilon = 0.1$, $\tau = 0$, and $\alpha = 1$ (see Definition 2.4.3).

Weak Scaling. We measured the weak scalability of all phases of computation (the distributed tree building, the tree walk, and the computation). The data distribution we consider is a set of uniformly distributed 10-dimensional points. We vary the number of cores from 96 to 6144, adding 166,912 points per core. We used $\epsilon = 0.1$ and decreased the bandwidth parameter h as more cores are added to keep the number of distance computations constant per core; a similar experiment setup was used in [162], though we plan to perform more thorough evaluations. The timings for the computation maintains around 60 % parallel efficiency above 96 cores.

Strong Scaling. Figure 60 presents strong scaling results on a 10 million/4-dimensional subset of the SDSS dataset. We used the Epanechnikov kernel with $h = 0.000030518$

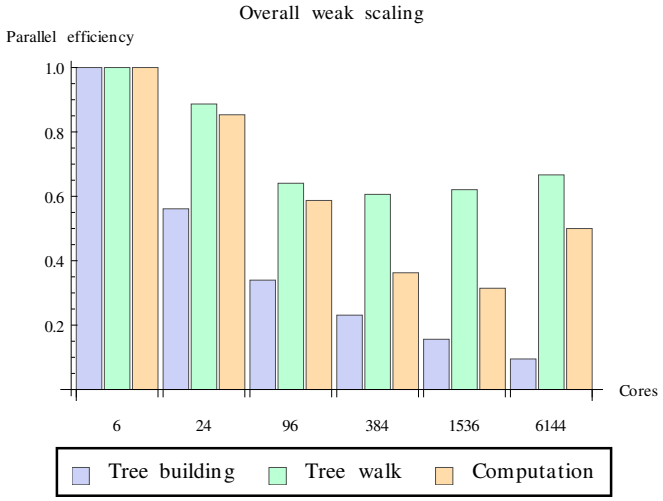


Figure 59: Weak scaling result for overall kernel summation computation on an uniform point distribution in 10 dimensions. We used 166,912 points / core and $\epsilon = 0.1$ and $h = \frac{1}{1024}$, halving h for every 4-fold increase in the number of cores. The base-timings for 6 cores are: 2.84 seconds for tree building, 1.8 seconds for the tree walk, and 128 seconds for the computation. The raw timings for all triples are (in seconds): (2.84, 1.8, 128), (5.06, 2.03, 150), (8.36, 2.81, 218), (12.3, 2.97, 353), (18.2, 2.9, 407), (29.9, 2.7, 258).

(chosen by the plug-in rule) with $\epsilon = 0.1$.

8.5 Conclusion

In this paper, we proposed a hybrid MPI/OpenMP kernel summation framework for scaling many popular data analysis methods. Our approach has advantages including: 1) the platform-independent C++ code base that utilize standard protocols such as MPI and OpenMP; 2) the template code structure that uses any multidimensional binary trees and any approximation schemes that may be suitable for high-dimensional problems; 3) extendibility to a large class of problems that require fast evaluations of kernel sums. Our future work will address: 1) distributed computation on unreliable network connections; 2) extending to take advantage of heterogeneous architectures including GPGPUs for a hybrid MPI/OpenMP/CUDA framework; 3) extension of the parallel engine to handle problems with more than pair-wise interactions, such as the computation of n-point correlation functions [78, 132].

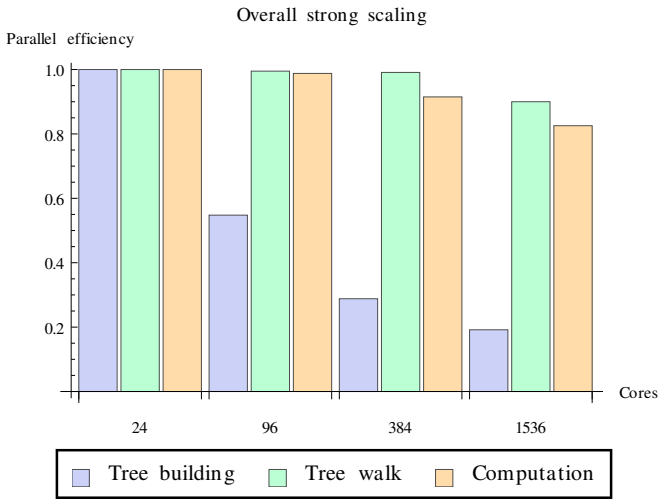


Figure 60: Strong scaling result for overall kernel summation computation on the 10 million subset of SDSS Data Release 6. The base timings for 24 cores are: 13.5 seconds, 340 seconds, 2370 seconds for tree building, tree walk, and computation respectively. The raw timings for all triples are (in seconds): (13.52, 339.36, 2371), (7.41, 24.38, 244), (2.93, 2.78, 98.78), (1.10, 0.27, 39.51).

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

CHAPTER IX

DISTRIBUTED AVERAGING AND RANDOM FEATURES FOR LARGE SCALE ANALYSIS

In this chapter, we discuss how to combine the distributed averaging algorithm with the random feature extraction method (see Section 2.3.4) for fast distributed (parallel) kernel eigendecomposition and inversion.

We assume there are a set of processes in a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{1, \cdots, N\}$ represents the nodes. and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ represents the pairs $(i, j) \in \mathcal{E}$ that can communicate directly. The set of neighbors of node i is denoted as $\mathcal{N}_i = \{j \in \mathcal{V} : (i, j) \in \mathcal{E}\}$ and its degree as $d_i = |\mathcal{N}_i|$.

9.1 Distributed Averaging

We first start by introducing the distributed averaging algorithm. We assume each node has a measurement $u_i \in \mathbb{R}$ and needs to compute the global average $\bar{u} = \frac{1}{N} \sum_{i=1}^{N} u_i$. The easiest way is to invoke a global communication primitive such as all-reduce MPI operation. However, this requires a formation of a spanning tree where a single process acts as the master. Each invocation requires all processes to synchronize and can cause bottlenecks in scalability. In addition, all-reduce operation can be useless in the cases of imperfect communication and dynamic network topologies. Distributed averaging can be used to circumvent this process at the expense of taking more number of iterations to converge to the true global average.

[199] considers the problem of finding a linear iteration yielding distributed averaging consensus over a network. In this chapter, we use a simple linear iteration called *average consensus algorithm*. Each process maintains an average x_i initialized with

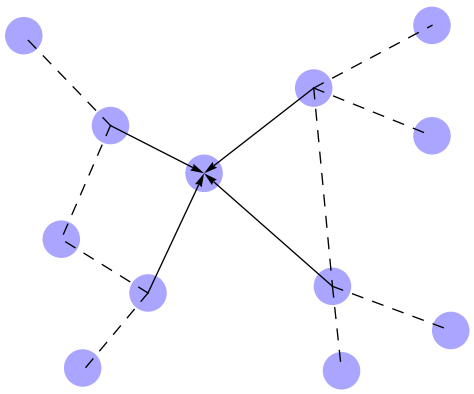


Figure 61: A set of interconnected processes where each connection represents the capability to communicate. Each synchronization phase requires each process to exchange with its immediate neighbors. The convergence is guaranteed as long as the network is a connected graph.

its own local number $x_i(0) = u_i$. Each process then iterates the difference equation by using the difference equation:

$$x_i(t+1) = x_i(t) + \epsilon \sum_{j \in \mathcal{N}_i} (x_j(t) - x_i(t)) \tag{9.1.1}$$

where $\epsilon < \frac{1}{\max\limits_{1 \leq i \leq N} d_i}$. Note that there are non-linear variants of difference equations [102]. Extensions to multivariate data and multiple data point cases are trivial [188].

It can be shown that each state converges to the average of the initial values on all nodes, that is $\lim\limits_{t \leftarrow \infty} x_i = \bar{u}$, as long as the graph $\mathcal{G}$ is connected. The all-reduce MPI operation can be thought of as a special case of distributed averaging where every process is in a fully connected network and communicated with the other processes in each synchronization phase. On the other hand, distributed averaging requires a smaller volume of communication and localized synchronization points. We note that [188] applied the distributed averaging for developing distributed computer vision algorithm such as point triangulation, linear pose estimation and affine structure from motion; the distributed linear algebra algorithms such as SVD, nullspace estimation, linear least squares, PCA, and generalized PCA are used as building blocks.

9.2 Kernel Matrix Inversion

Now we tackle the problem of inverting a large kernel matrix $\mathbf{K} = \{k(\mathbf{r_i}, \mathbf{r_j})\}_{\mathbf{r_i}, \mathbf{r_j} \in \mathbf{R}}$ (see Figure 5), that is, the computation of $\mathbf{K}^{-1}\mathbf{y}$. We note that [189] has already applied distributed averaging for this case and briefly summarize their derivations. We then propose to make a small change in their algorithm by introducing random feature extraction.

9.2.1 Gaussian Process Regression

We assume the following regression model $\hat{y} = f(\mathbf{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. We briefly introduce the Gaussian process regression model, where $f(\mathbf{x})$ is a zero-mean Gaussian process with covariance function $K(\mathbf{x}, \mathbf{x}'') : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution [153]. It is completely specified by a mean function $m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$ (typically assumed to be zero) and a covariance function $K(\mathbf{x}, \mathbf{x}'') = \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}'') - m(\mathbf{x}''))] = \mathbb{E}[f(\mathbf{x})f(\mathbf{x}'')]$.

If we have training data $\mathbf{D} = \begin{bmatrix} \mathbf{x_1}, & \cdots, & \mathbf{x_N} \\ y_1, & \cdots, & y_N \end{bmatrix}$, the $N \times N$ covariance matrix $\mathbf{K}$ is now defined as $[\mathbf{K}]_{jk} = K(\mathbf{x_j}, \mathbf{x_k})$. We then define the observation vector $\mathbf{y} = [y_1, \ldots, y_N]^T$; $\mathbf{y}$ can be shown as a zero mean multivariate Gaussian process with a covariance matrix $\mathbf{K}^* = \mathbf{K} + \sigma^2\mathbf{I}$. The posterior density for a test point $\mathbf{x}^*$, $p(y^*|\mathbf{x}^*, \mathbf{D})$ is a univariate normal distribution with the mean $\bar{y}^*$ and the variance $\text{var}(y^*)$:

$$\bar{y}^* = \mathbf{k}(\mathbf{x}^*)^T(\mathbf{K}^*)^{-1}\mathbf{y}$$

$$\text{var}(y^*) = K(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}(\mathbf{x}^*)^T(\mathbf{K}^*)^{-1}\mathbf{k}(\mathbf{x}^*)$$

where $\mathbf{k}(\mathbf{x}^*) = [k(\mathbf{x}^*, \mathbf{x_1}), \ldots, k(\mathbf{x}^*, \mathbf{x_n})]^T$. We focus on Gaussian Automatic Relevance Determination (ARD) kernel as the covariance function:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp\left[-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T diag\left(\frac{1}{\theta_1^2}, \frac{1}{\theta_2^2}, \cdots, \frac{1}{\theta_D^2}\right)(\mathbf{x} - \mathbf{x}')\right] \tag{9.2.1}$$

For simplicity, we further restrict to the case of the Gaussian kernel ($h = \theta_1 = \cdots = \theta_D$, $\sigma_f = 1$). Nevertheless, we note that our discussions can be trivially extended to the Gaussian ARD kernel and more importantly all positive-definite kernels with well-defined inverse Fourier transforms.

A common choice of recovering f is to minimize the following cost function:

$$Q(f) = \sum_{i=1}^{N} (y_i - f(\mathbf{x_i}))^2 + \gamma ||f||^2_{\mathcal{H}_K} \tag{9.2.2}$$

where $\mathcal{H}_K$ is the hypothesis space corresponding to a reproducing kernel Hilbert space (RKHS) defined by the kernel $k(\cdot, \cdot)$. γ is the regularization parameter trading empirical evidence and smoothness of f. It can be shown that $f_c = \underset{f \in \mathcal{H}_K}{\arg\min}\, Q(f)$ satisfies the following properties:

$$f_c = \sum_{i=1}^{N} c_i k(\mathbf{x_i}, \cdot), \qquad \begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix} = (\mathbf{K} + \gamma \mathbf{I})^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Now we formally define the computational problem.

Problem: Given the set of reference point and target pairs $\mathbf{D} = \begin{bmatrix} \mathbf{x_1}, & \cdots, & \mathbf{x_N} \\ y_1, & \cdots, & y_N \end{bmatrix}$, the positive definite kernel k, and the regularization parameter $\gamma > 0$,

Task: Compute $\begin{bmatrix} c_1 \\ \vdots \\ c_N \end{bmatrix} = (\mathbf{K} + \gamma \mathbf{I})^{-1} \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$ as fast as possible.

9.2.2 Applying Distributed Averaging

[189] restricts the hypothesis space to a closed subspace $\breve{\mathcal{H}}_K \subset \mathcal{H}$ by using the eigenexpansion of the kernel k. An eigenfunction $\phi(\cdot) : \mathbb{R}^D \leftarrow \mathbb{R}$ obeys the following integral equation with respect to measure μ:

$$\int k(\mathbf{x}, \mathbf{x}') \phi(x) \partial \mu(\mathbf{x}) = \lambda \phi(\mathbf{x}') \tag{9.2.3}$$

Generally, there are an infinite number of eigenfunctions, $\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \ldots$, and corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots$. Eigenfunctions are orthogonal with respect to μ and normalized such that $\int \phi_i(\mathbf{x})\phi_j(\mathbf{x})\partial\mu(\mathbf{x}) = \delta_{ij}$ where δ_{ij} is the Kronecker delta. By Mercer's theorem, we can expand:

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i^*(\mathbf{x}') \tag{9.2.4}$$

[189] then truncates Equation (9.2.4) after d terms. The new restricted hypothesis subspace precisely is the following:

$$\breve{\mathcal{H}}_K = \overline{span\{\phi_1, \cdots, \phi_d\}} \tag{9.2.5}$$

For each point $\mathbf{x}$ in the query/reference datasets, a set of new features is generated using the first d eigenfunctions: $[\phi_1(\mathbf{x}), \cdots, \phi_d(\mathbf{x})]$. Now we instead get the following function:

$$\hat{f}_S = \underset{f \in \breve{\mathcal{H}}_K}{\arg\min} \sum_{i=1}^{N} (y_i - f(\mathbf{x_i}))^2 + \gamma ||f||^2_{\mathcal{H}_K} \tag{9.2.6}$$

See Proposition 5 in [189] for the goodness of the estimator $\hat{f}_S$.

We note that Gaussian process regression is an infinite-dimensional kernelized ridge regression (Chapter 2 in [153]). In essence, we use the eigenfunctions to generate a low-dimensional (finite) subspace so that linear methods can be applied [150]. Therefore, we can write $\hat{f}_S$ in the following way:

$$\hat{f}_S(\mathbf{x}) = \mathbf{g}(\mathbf{x})^T \left(\mathbf{G}\mathbf{G}^T + diag\left(\frac{\gamma}{\lambda_1}, \cdots, \frac{\gamma}{\lambda_d} \right) \right)^{-1} \mathbf{s} \tag{9.2.7}$$

where $\mathbf{g}(\mathbf{x}) = \begin{bmatrix} \phi_1(\mathbf{x}) & \cdots & \phi_d(\mathbf{x}) \end{bmatrix}^T$, $\mathbf{G} = \begin{bmatrix} \mathbf{g}(\mathbf{x_1}) & \cdots & \mathbf{g}(\mathbf{x_N}) \end{bmatrix}$, and $\mathbf{s} = \sum_{i=1}^{N} \mathbf{g}(\mathbf{x_i}) y_i$. Now note that $\frac{1}{N}\mathbf{G}\mathbf{G}^T = \frac{1}{N}\sum_{i=1}^{N} \mathbf{g}(\mathbf{x_i})\mathbf{g}(\mathbf{x_i})^T$ and $\frac{1}{N}\mathbf{s} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{g}(\mathbf{x_i}) y_i$ are global averages and distributed averaging can be applied.

Instead of using the eigenfunctions $\phi_1, \cdots, \phi_d$, we can also expand the kernel using its inverse Fourier transform (Equation (2.3.2)) and randomly sample $\boldsymbol{\xi}_{\boldsymbol{\omega}}(\mathbf{x}) =$

$\left[\cos(\boldsymbol{\omega}^T\mathbf{x}), \sin(\boldsymbol{\omega}^T\mathbf{x})\right]^T$ such that $k(\mathbf{x}, \mathbf{x}') \approx \frac{1}{d}\sum_{i=1}^{d} \boldsymbol{\xi}_{\boldsymbol{\omega}_i}(\mathbf{x})^T \boldsymbol{\xi}_{\boldsymbol{\omega}_i}(\mathbf{x}')$. Now we can re-write Equation (9.2.7):

$$\hat{f}_R(\mathbf{x}) = \boldsymbol{\xi}(\mathbf{x})^T \left(\boldsymbol{\Xi}\boldsymbol{\Xi}^T + \gamma\mathbf{I}\right)^{-1} \mathbf{s}_{\boldsymbol{\Xi}} \tag{9.2.8}$$

where $\boldsymbol{\xi}(\mathbf{x}) = \frac{1}{\sqrt{d}}\left[\boldsymbol{\xi}_{\boldsymbol{\omega_1}}(\mathbf{x})^T \quad \cdots \quad \boldsymbol{\xi}_{\boldsymbol{\omega_d}}(\mathbf{x})^T\right]^T$, $\boldsymbol{\Xi} = \left[\boldsymbol{\xi}(\mathbf{x_1}) \quad \cdots \quad \boldsymbol{\xi}(\mathbf{x_N})\right]$, and $\mathbf{s}_{\boldsymbol{\Xi}} = \sum_{i=1}^{N} \boldsymbol{\xi}(\mathbf{x_i}) y_i$. $\hat{f}_R$ is a solution in another hypothesis subspace:

$$\widetilde{\mathcal{H}}_K = \overline{span\{\boldsymbol{\xi}_{\boldsymbol{\omega_1}}, \cdots, \boldsymbol{\xi}_{\boldsymbol{\omega_d}}\}} \tag{9.2.9}$$

Compared to the eigenexpansion case, we can no longer reduce the cost of each communication between a pair of processes to $\mathcal{O}(d)$; this was possible due to the orthogonality of eigenfunctions and the replacement of the covariance term $\frac{1}{N}\mathbf{G}\mathbf{G}^T$ with its expected value $\mathbb{E}_\mu\left[\mathbf{G}\mathbf{G}^T\right] = \mathbf{I}$, i.e. the identity matrix (see Section 4.2 of [189]).

9.3 Kernel Eigendecomposition

9.3.1 Kernel PCA

We briefly introduce one of the widely used feature extraction method called kernel PCA [164] in this section[1]. Many dimensionality reduction methods such as FastMap [63], IsoMap [186], Local Linear Embedding [157], Local Tangent Space Alignment [208], Multidimensional scaling [108], Laplacian eigenmaps [13] can be cast as a special case of Kernel PCA [91, 196, 165, 14].

The kernel in the context of kernel PCA (and other kernel methods [165]) uses $k(\cdot, \cdot)$ as a way of expressing dot products in the feature space $\mathcal{F}$, that is $k(\mathbf{x}, \mathbf{y}) = \boldsymbol{\Phi}(\mathbf{x})^T\boldsymbol{\Phi}(\mathbf{y})$ for $\Phi : \mathbb{R}^D \to \mathcal{F}$. Kernel PCA is used for extracting principal components in $\mathcal{F}$ rather than the original input space $\mathbb{R}^D$ for discovering latent structures in the data.

[1]This is an unfortunate overloading of terms which may cause confusion. Any suitable probability density function can be used as a kernel in non-parametric density estimation, whereas kernels in kernel PCA have different requirements.

We denote the empirical average of the feature map as $\boldsymbol{\mu}_{\mathbf{R}} = \frac{1}{|\mathbf{R}|} \sum_{\mathbf{r_j} \in \mathbf{R}} \boldsymbol{\Phi}(\mathbf{r_j})$. We define the covariance matrix in $\mathbf{F}$, $\bar{\mathbf{C}} = \frac{1}{|\mathbf{R}|} \sum_{\mathbf{r_i} \in \mathbf{R}} (\boldsymbol{\Phi}(\mathbf{r_i}) - \boldsymbol{\mu}_{\mathbf{R}}) (\boldsymbol{\Phi}(\mathbf{r_i}) - \boldsymbol{\mu}_{\mathbf{R}})^T$. The principal components of $\boldsymbol{\Phi}(\mathbf{R}) = \{\boldsymbol{\Phi}(\mathbf{r_j})\}_{\mathbf{r_j} \in \mathbf{R}}$ are given by the eigenvectors of $\bar{\mathbf{C}}$. The vanilla PCA corresponds to the case of $\boldsymbol{\Phi}(\mathbf{x}) = \mathbf{x}$. Because $\boldsymbol{\Phi}(\cdot) : \mathbb{R}^D \to \mathbf{F}$ could be an infinite-dimensional mapping (i.e. $\boldsymbol{\Phi}(\cdot)$ induced by the Gaussian kernel), we instead eigendecompose the centered kernel matrix: $\bar{\mathbf{K}} = \mathbf{V}\boldsymbol{\Sigma}^2 V^T$, where

$$\begin{aligned}\bar{\mathbf{K}}_{i,j} &= (\boldsymbol{\Phi}(\mathbf{r_i}) - \boldsymbol{\mu}_{\mathbf{R}})^T (\boldsymbol{\Phi}(\mathbf{r_j}) - \boldsymbol{\mu}_{\mathbf{R}}) \\ &= k(\mathbf{r_i}, \mathbf{r_j}) - \boldsymbol{\Phi}(\mathbf{r_i})^T \boldsymbol{\mu}_{\mathbf{R}} - \boldsymbol{\Phi}(\mathbf{r_j})^T \boldsymbol{\mu}_{\mathbf{R}} + \boldsymbol{\mu}_{\mathbf{R}}^T \boldsymbol{\mu}_{\mathbf{R}}\end{aligned}$$

and for any $\mathbf{x} \in \mathbb{R}^D$, $\boldsymbol{\Phi}(\mathbf{x})^T \boldsymbol{\mu}_{\mathbf{R}} = \frac{1}{\mathbf{R}} \sum_{\mathbf{r_i} \in \mathbf{R}} k(\mathbf{x}, \mathbf{r_i})$ and $\boldsymbol{\mu}_{\mathbf{R}}^T \boldsymbol{\mu}_{\mathbf{R}} = \frac{1}{|\mathbf{R}|^2} \sum_{\mathbf{r_m}, \mathbf{r_p} \in \mathbf{R}} k(\mathbf{r_m}, \mathbf{r_p})$. The overall computation problem here is defined as the following.

Problem: Given the set of reference point and target pairs $\mathbf{R} = \begin{bmatrix} \mathbf{x_1}, & \cdots, & \mathbf{x_N} \end{bmatrix}$, the positive definite kernel k,

Task: Compute the top m eigenvectors/eigenvalues of the kernel matrix $\bar{K}$ as fast as possible.

9.3.2 Previous Approaches

Now we go back to the problem of eigendecomposing a large kernel matrix $\mathbf{K}$ (see Figure 4). A common operation in data analysis is an eigendecomposition of the kernel matrix $\mathbf{K} = \{k(\mathbf{r_i}, \mathbf{r_j})\}_{\mathbf{r_i}, \mathbf{r_j} \in \mathbf{R}}$. Here we discuss previous approaches for eigendecomposing large kernel matrices in data analysis. A naive method requires $\mathcal{O}(|\mathbf{R}|^2)$ storage and $\mathcal{O}(|\mathbf{R}|^3)$ computational cost.

Deterministic Approximations. [66] pointed out that the incomplete Cholesky factorization to compute a low rank approximation $\widetilde{\mathbf{K}} = \mathbf{G}\mathbf{G}^T$ such that $\widetilde{\mathbf{K}} \simeq \mathbf{K}$. However, this method is not matrix-free since it works on the precomputed $\mathbf{K}$. [178] proposes a distributed algorithm for spectral clustering which involves an eigendecomposition of the Laplacian of $\mathbf{K}$. It distributes the rows of sparsified $\mathbf{K}$ equally

among p MPI processes and employs a parallel eigensolver.

Probabilistic Approximations. [195] uses the Nystrom method to reduce the cubic computational cost by carrying out an eigendecomposition on a smaller kernel matrix formed from the random subset of $\mathbf{R}$. The original eigendecomposition problem is recovered by the Nystrom extension formula. A similar idea is used in [61], which instead uses a probabilistic row or column sampling of the precomputed $\mathbf{K}$.

[2] proposes to replace $\mathbf{K}$ with its randomized variant using random projections and randomized rounding. This is motivated by the fact that classical linear algebra approaches such as orthogonal/Lanczos iterations (which are sometimes used to compute KPCA) have a huge computational bottleneck in matrix-matrix product that occurs inside each iteration. Randomized rounding sparsifies $\mathbf{K}$, which helps accelerate the matrix-matrix product inside orthogonal/Lanczos iteration, while random projection reduces the dimensionality of each point (classical Johnson-Lindenstrauss lemma states that there exists a lower dimensional embedding that preserves the pairwise distance with very small error), meaning the pairwise distances are computed much faster.

[62] proposes a distributed algorithm of [2]. Each process receives a randomly sparsified version of $\mathbf{K}$ and computes its own eigenvectors, and the master process gathers and averages them. However, the authors assume that $\mathbf{K}$ is stored on the master process.

9.3.3 Distributed Averaging-based Approach

Following the derivations in [189] and Section 9.2.2, we can develop a distributed averaging scheme for large-scale kernel PCA. Note that we need to compute $\mathbf{K} \approx \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T$ where $\mathbf{U} \in \mathbb{R}^{N\times r}$, $\mathbf{\Sigma} \in \mathbb{R}^{r\times r}$, and r is the desired rank. Algorithm 9.3.1 shows the serial version of the random-feature-based (non-centered) kernel PCA algorithm.

Algorithm 9.3.1 Random feature-based (non-centered) kernel PCA algorithm ($\frac{1}{N}\mathbf{K} = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T$).

Compute the random projection of the dataset $\mathbf{\Xi} = \begin{bmatrix}\boldsymbol{\xi}(\mathbf{x_1}) & \cdots & \boldsymbol{\xi}(\mathbf{x_N})\end{bmatrix}$
Compute the covariance matrix $\mathbf{C} = \frac{1}{N}\mathbf{\Xi}\mathbf{\Xi}^T$.
Eigendecompose $\mathbf{C} = \frac{1}{N}\mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^T$.
Compute $\mathbf{U} = \mathbf{G}^T\mathbf{V}\mathbf{\Sigma}^{-1}$.

Parallel Eigendecomposition. Note that the covariance $\mathbf{C} = \frac{1}{N}\mathbf{\Xi}\mathbf{\Xi}^T = \sum_{i=1}^{N} \boldsymbol{\xi}(\mathbf{x_i})\boldsymbol{\xi}(\mathbf{x_i})^T$ is a global average. If $\mathbf{\Xi} = \begin{bmatrix}\mathbf{\Xi}_1 & \cdots & \mathbf{\Xi}_p\end{bmatrix}$ are distributed on multiple processes (where P-th process owns the block matrix $\mathbf{\Xi}_P$), then we can simply compute local covariances (which are averages) and apply the distributed averaging algorithm. Once each process has its own local estimate of the covariance $\mathbf{C}_{P,local}$ after a number of iterations of distributed averaging, it can compute the eigendecomposition of $\mathbf{C}_{P,local} = \frac{1}{N}\mathbf{V}_{P,local}\mathbf{\Sigma}^2_{P,local}\mathbf{V}^T_{P,local}$. [2] Note that $\mathbf{U} = \begin{bmatrix}\mathbf{U}_1 \\ \vdots \\ \mathbf{U}_p\end{bmatrix} = \mathbf{\Xi}^T V\mathbf{\Sigma}^{-1}$ where $\mathbf{U}_P \in \mathbb{R}^{N_P \times r}$ is the eigenvector components owned by the P-th process and N_P is the number of reference points owned by the P-th process. Each P-th process can now compute its own eigenvector components: $\mathbf{U}_{P,local} = \mathbf{\Xi}_P^T\mathbf{V}_{P,local}\mathbf{\Sigma}^{-1}_{P,local}$.

KPCA Predictions. Given the n-th eigenvector of $\bar{\mathbf{K}}$, $\boldsymbol{\alpha}_n \in \mathbb{R}^{|\mathbf{R}|}$, the n-th eigenvector of $\bar{\mathbf{C}}$ is $\mathbf{V}_n = \sum_{\mathbf{r_i}\in\mathbf{R}} \boldsymbol{\alpha}_n[i]\left(\mathbf{\Phi}(\mathbf{r_i}) - \boldsymbol{\mu}_\mathbf{R}\right)$. The n-th kernel principal component of a given test point $\mathbf{q}$ is:

$$\begin{aligned}
&(KPC)_n(\mathbf{q}) \\
&= \left(\mathbf{\Phi}(\mathbf{q}) - \boldsymbol{\mu}_\mathbf{R}\right)^T \mathbf{V}_n \\
&= \sum_{\mathbf{r_i}\in\mathbf{R}} \boldsymbol{\alpha}_n[i]\left(\mathbf{\Phi}(\mathbf{q}) - \boldsymbol{\mu}_\mathbf{R}\right)^T\left(\mathbf{\Phi}(\mathbf{r_i}) - \boldsymbol{\mu}_\mathbf{R}\right) \\
&= \sum_{\mathbf{r_i}\in\mathbf{R}} \boldsymbol{\alpha}_n[i]\left(k(\mathbf{q},\mathbf{r_i}) - \mathbf{\Phi}(\mathbf{r_i})^T\boldsymbol{\mu}_\mathbf{R} - \mathbf{\Phi}(\mathbf{q})^T\boldsymbol{\mu}_\mathbf{R} + \boldsymbol{\mu}_\mathbf{R}^T\boldsymbol{\mu}_\mathbf{R}\right) \qquad (9.3.1)
\end{aligned}$$

For a test point $\mathbf{q} \in \mathbf{Q}$, computing the n-th kernel principal components requires:

[2] It is possible to adapt this to the case where N is not known or hard to find.

- The weighted kernel sum, $\sum_{\mathbf{r_i} \in \mathbf{R}} \boldsymbol{\alpha}_n[i] k(\mathbf{q}, \mathbf{r_i})$.
- The average kernel value for $\mathbf{q}$, $\boldsymbol{\Phi}(\mathbf{q})^T \boldsymbol{\mu}_{\mathbf{R}}$.
- The dot product between $\boldsymbol{\alpha}_n$ and the vector of average kernel values $[\boldsymbol{\Phi}(\mathbf{r_1})^T \boldsymbol{\mu}_{\mathbf{R}}, \cdots, \boldsymbol{\Phi}(\mathbf{r}_{|\mathbf{R}|})^T \boldsymbol{\mu}_{\mathbf{R}}]^T$.
- The average kernel value among the training set $\boldsymbol{\mu}_{\mathbf{R}}^T \boldsymbol{\mu}_{\mathbf{R}}$.

Note that the vector of average kernel values for the test set $[\boldsymbol{\Phi}(q_1)^T \mu_{\mathbf{R}}, \cdots, \boldsymbol{\Phi}(q_{|\mathbf{Q}|})^T \mu_{\mathbf{R}}]$ and for the training set $[\boldsymbol{\Phi}(\mathbf{r_1})^T \boldsymbol{\mu}_{\mathbf{R}}, \cdots, \boldsymbol{\Phi}(\mathbf{r}_{|\mathbf{R}|})^T \boldsymbol{\mu}_{\mathbf{R}}]$ and the average kernel value $\boldsymbol{\mu}_{\mathbf{R}}^T \boldsymbol{\mu}_{\mathbf{R}}$ can be computed once. The computational cost of KPCA projection for a set of test points $\mathbf{Q}$ is naively $\mathcal{O}(D(|\mathbf{Q}||\mathbf{R}| + |\mathbf{R}|^2))$. All of the above operations can accelerated using the techniques illustrated in Chapter 3, Chapter 4, Chapter 5, and Chapter 8.

9.4 Experiments

In this section, we focus on the two items that have not been addressed in Table 13 of Chapter 8. As mentioned earlier, we restrict ourselves to the case of positive definite kernels with well-defined inverse Fourier transforms so that the random feature extraction can be applied. Threads were used to parallelize 1) the random feature-based projection step; 2) the computation of required local averages.

9.4.1 Kernel Matrix Inversion: Gaussian Process Regression

Here we test on the 4-dimensional subset of the SDSS (Sloan Digital Sky Survey) Dataset Release 6 containing 39,761,242 points as our training set, a larger dataset than the one used in Chapter 8; we took the first three dimensions for predicting the last dimension and scaled the first three dimensions to fit in the unit hypercube $[0, 1]^3$. We ran our experiments on the Hopper cluster utilizing 384 cores (16 nodes, 24 cores/node, 4 MPI processes/node, 6 threads/MPI process). We produced the test

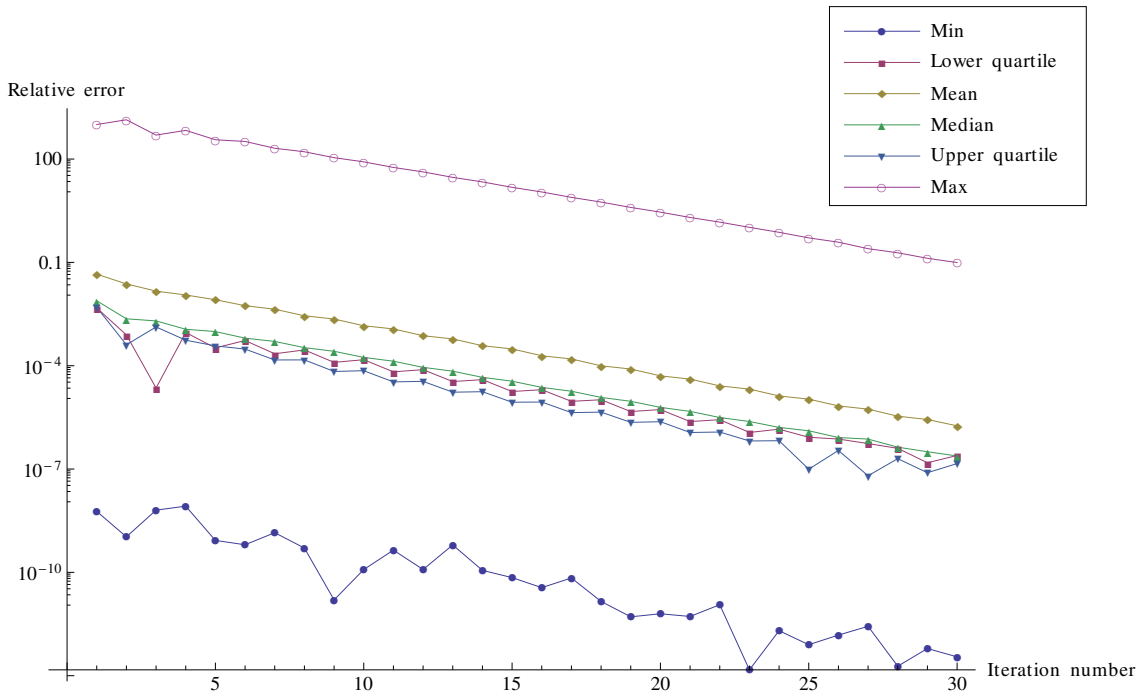


Figure 62: The plot of the minimum, lower quartile, mean, median, upper quartile, and maximum relative errors among the predicted query regression estimates in each iteration across all MPI processes.

set of the same size as the training set by adding Gaussian noises to the training set. We measure the relative error deviation between the predicted centralized estimates provided by the random features and the predicted localized estimated provided by the distributed averaged random features across all MPI processes (see Figure 62). We used the Gaussian kernel with the bandwidth that maximizes the marginal loglikelihood on a randomly chosen 10,000 reference points. In each iteration of distributed averaging we used, each process P communicates at most $\log_2 p$ neighbors in the hypercube network topology.

9.4.2 Kernel Matrix Eigendecomposition: Kernel PCA

Here we test on the MNIST dataset [111] containing 60,000 points. Each point represents a 28×28 image (784 dimensions). We measure the scalability of distributed averaging-based approach stopping after the first 10 iterations. This experiment was

performed on the Lincoln cluster (retired and replaced by the Forge cluster as of August 15, 2011) at XSEDE; the Lincoln Cluster consisted of 192 compute nodes and 96 NVIDIA Tesla units (which we do not utilize in this section). Given the 60,000 points, we generated a larger dataset by perturbing the original dataset by adding Gaussian noises. We test on the four different configurations: flat MPI, 2 threads/MPI process, 4 threads/MPI process, and 8 threads/MPI process. Figure 63 shows the scalability results.

9.5 Conclusion

In this chapter, we explored a way of combining the random feature extraction method with the distributed averaging framework to scale the training phases of two kernel methods: Gaussian process regression and kernel PCA. For Gaussian process regression, the problem of inverting the kernel matrix can be reduced to the problem of computing the ridge regression solution of the random-feature linearized problem. For kernel PCA, the problem of eigendecomposing the kernel matrix can be reduced to the problem of computing the eigendecomposition of a smaller covariance matrix in the random feature space. If the communication is imperfect or slow, then the distributed averaging framework can be used to let each process exchange information locally. This provides the following advantages: 1) any-time estimate based on the local averages held by the given process; 2) the convergence guarantee to the centralized estimates; 3) the adaptability to the changes in the network topology.

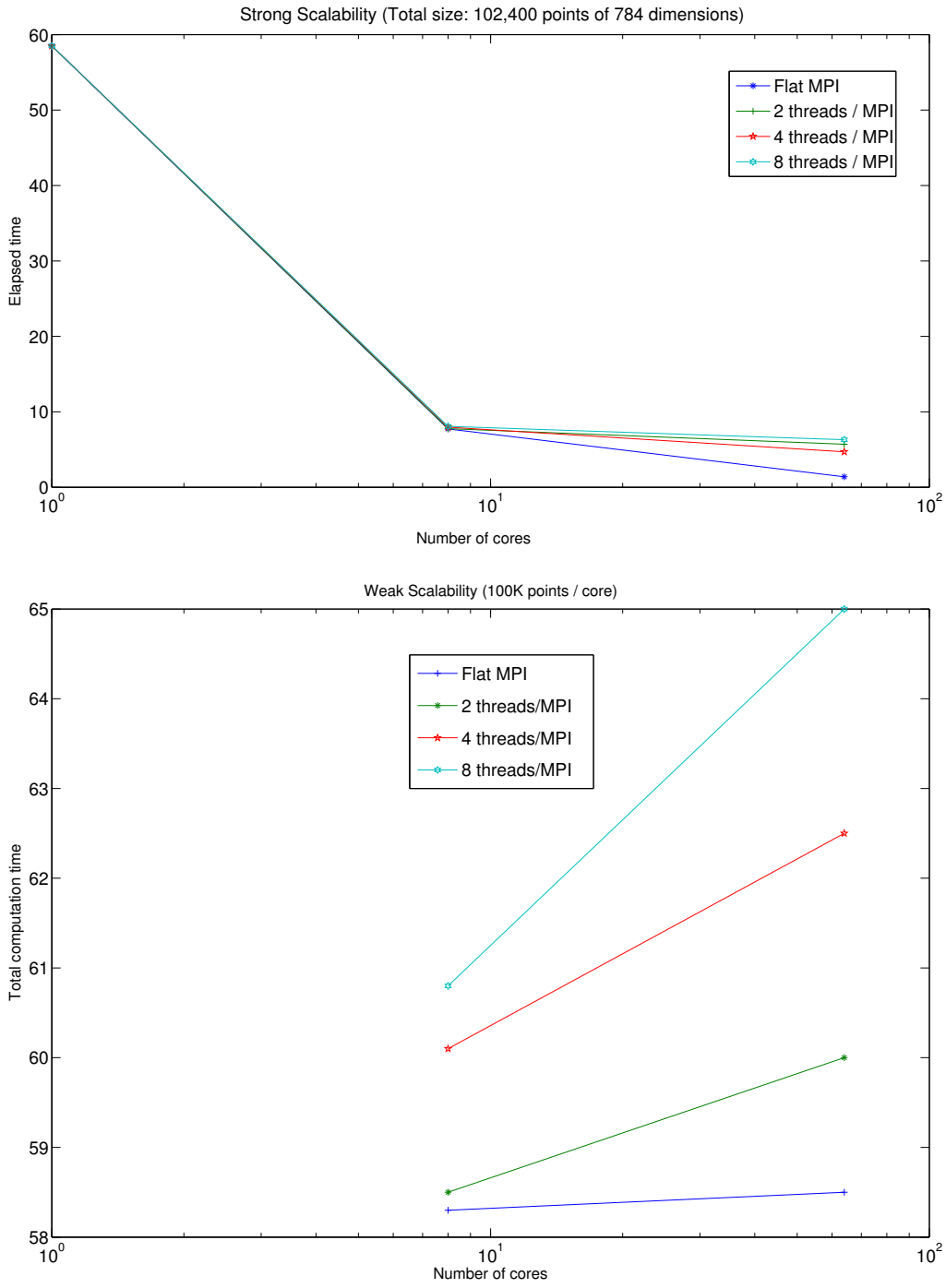


Figure 63: Top: the strong scalability experiment on 102,400 points up to 64 cores; **Bottom:** the weak scalability experiment adding 100,000 points/core.

CHAPTER X

CONCLUSION

In this thesis, we have explored various techniques for accelerating the computation of three fundamental linear algebraic operations ubiquitous in machine learning and scientific simulations. I claim there are three *fundamental linear algebraic operations* ubiquitous in many data mining and scientific algorithms:

1. Kernel summations: $\forall \mathbf{x_1} \in \mathbf{X_1}, \sum_{\mathbf{x_2} \in \mathbf{X_2}} k(\mathbf{x_1}, \mathbf{x_2})$
2. Solving a linear system involving a kernel matrix: $\mathbf{K}^{-1}\mathbf{y}$
3. Eigendecomposing matrices: $\mathbf{K} = \mathbf{U}\mathbf{\Sigma}\mathbf{U}^T$

The spectrum of the contributions of this thesis for tackling these problems can be attributed to various fields including:

1. Computational geometry: subspace tree (Chapter 5).
2. Computational physics: the first hierarchical fast Gauss transform (Chapter 3), a hierarchical fast Gauss transform with a different Cartesian expansion (Chapter 4), and its higher-order extension called multibody multipole method (Chapter 7).
3. High performance computing: parallel multidimensional tree building and handling the distributed data case (Chapter 8).
4. Distributed optimization: distributed averaging for fast kernel matrix inversion and kernel matrix eigendecomposition (Chapter 9).

Despite spanning these fields, this thesis demonstrates that many of these ideas are similar in nature and can be unified under a single purpose of accelerating the computation of kernel sums.

Future work include:

- **Parallelization of Multidimensional Tree Construction using GPGPU Parallelism:** Recent literature have focused on building the tree entirely on GPUs [209]. [27] is a recent work that builds the octree and runs the entire kernel summations within GPUs.

- **Incremental Update of Multidimensional Trees in Response to Changing Data Distribution:** Previous work includes [135] for kd-trees and [55] for random projection trees.

- **Investigation of Practical Data Structures for Statistical and Simulation Methods:** This work started during my years of undergraduate research for investigating practical data structures for the nearest-neighbor problem. A preliminary report [81] detailing empirical comparison of over forty different structures has been written. This study, however, was under the setting of single-core machines. I plan to do more thorough comparisons on heterogeneous/multicore architectures.

- **Distributed Optimization:** Alternating Direction Method of Multipliers [23] is gaining attention in the machine learning community.

- **Graphical Model Inference and Efficient Implementations:** Kernel methods and graphical models constitute major parts in machine learning and data analysis. Whereas kernel methods model relationship between pairs of observations, graphical models can model relationship (i.e. conditional independence) among the attributes describing each multivariate observation. I plan to explore

relationship between graphical model inference methods and kernel methods to develop richer models. Because richer models come at the expense of requiring more computational time, I plan to delve into (1) parallelization/decomposition techniques for large-scale inference learning [131, 125] and (2) applying acceleration techniques for kernel methods developed in this thesis. In particular, [177] has applied random feature techniques for kernel methods in the context of accelerating a kernelized form of belief propagation, a fundamental operation in graphical model inference.

- **Large-Scale Real-Time Application:** Acceleration techniques investigated in my thesis have the potential to make real-time applications feasible, such as surveillance applications [103]. In [103], we have developed a new representation for matching motion trajectories using Gaussian process regression. Making this system feasible in a real application requires (1) an effective way of handling streaming data; (2) an efficient way of updating each representative model. I am planning to explore a GPGPU-based acceleration using the CUDA programming framework.

Publications that comprise and support this thesis include the following:

- Dongryeol Lee, Alexander G. Gray, and Andrew W. Moore. Dual-Tree Fast Gauss Transforms. In: Advances in Neural Information Processing Systems, 2005.

- Dongryeol Lee and Alexander G. Gray. Faster Gaussian Summation: Theory and Experiment. In: Proceedings of the Twenty-Second Conference on Uncertainty in Artificial Intelligence, 2006.

- Ping Wang, Dongryeol Lee, Alexander G. Gray, and James M. Rehg. Fast Mean Shift with Accurate and Stable Convergence. In: Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, 2007.

- Dongryeol Lee and Alexander G. Gray. Fast High-dimensional Kernel Summations Using the Monte Carlo Multipole Method, In: Advances in Neural Information Processing Systems, 2008.

- Dongryeol Lee, Alexander G. Gray, and Andrew W. Moore. Dual-Tree Fast Gauss Transforms (arXiv).

- Parikshit Ram, Dongryeol Lee, Hua Ouyang, and Alexander G. Gray. Rank-Approximate Nearest Neighbor Search: Retaining Meaning and Speed in High Dimensions. In: Advances in Neural Information Processing Systems, 2009.

- Parikshit Ram, Dongryeol Lee, William B. March, and Alexander G. Gray. Linear-time Algorithms for Pairwise Statistical Problems. In: Advances in Neural Information Processing Systems, 2009. **Spotlight Presentation**.

- Dongryeol Lee, Arkadas Ozakin, and Alexander G. Gray. Multibody Multipole Methods. Under submission to Journal of Computational Physics, 2011.

- Kihwan Kim, Dongryeol Lee, and Irfan Essa. Gaussian Process Regression Flow for Analysis of Motion Trajectories. In: Proceedings of IEEE International Conference on Computer Vision, 2011.

- William B. March, Arkadas Ozakin, Dongryeol Lee, Ryan Riegel, and Alexander G. Gray. Multi-Tree Algorithms for Large-Scale Astrostatistics. In Advances in Machine Learning and Data Mining for Astronomy, Chapman and Hall/CRC Press, 2012.

- Dongryeol Lee, Richard Vuduc, and Alexander G. Gray. A Distributed Kernel Summation Framework for General-Dimension Machine Learning. To appear in SIAM International Conference on Data Mining, 2012. **Best Paper Award**.

- Parikshit Ram, Dongryeol Lee, and Alexander G. Gray. Nearest-Neighbor Search on a Time Budget via Max-Margin Trees. To appear in SIAM International Conference on Data Mining, 2012.

- Kihwan Kim, Dongryeol Lee, and Irfan Essa. Detecting Regions of Interest in Dynamic Scenes for Camera Motion. To appear in IEEE Conference on Computer Vision and Pattern Recognition, 2012.

APPENDIX A

PSEUDOCODE FOR SERIES EXPANSION

This section explains how to implement the $\mathcal{O}(p^D)$ series-expansion mechanisms in computer languages such as C/C++.

Storing the Far-field/Local Moments as a Linear Array. Although the moments are inherently multi-dimensional, we store all coefficients in a C-style one-dimensional array. Each query node stores $(p_{max}+1)^D$ local moment terms. Similarly, each reference node stores $(p_{max}+1)^D$ far-field moment terms. These are allocated as a linear array during the construction of the two trees, as shown in Figure 26 which implies a bijective mapping between D-digit radix-$(p_{max}+1)$ numbers and decimal numbers between 0 and $(p_{max}+1)^D$ - 1 inclusive.

Converting between a Position and a Multi-index in the Linear Array. Algorithm A.0.1 shows the mapping from a position in the linear array of $(p_{max}+1)^D$ terms to its corresponding multi-index. The algorithm converts the given position (given in base 10) to a number in base p. Algorithm A.0.2 converts the given multi-

Algorithm A.0.1 PositionToMultiindex(i, p): Converts the position of a linear array of length $(p+1)^D$ to its multi-index.

```
{i-th position maps to the multi-index α.}
α_{i=1,···,D} ← 0
for d = D to d = 1 do
    α[d − (D − 1)] ← ⌊ i/(p+1) ⌋
    i ← i mod (p + 1)
return α
```

index to its corresponding position in the linear array of length $(p_{max}+1)^D$.

Algorithm A.0.2 MULTIINDEXTOPOSITION($\boldsymbol{\alpha}$): Converts the given multi-index to its corresponding position in the linear array of length $(p_{max}+1)^D$.

$x \leftarrow 0, \qquad f \leftarrow 1$
for $d = D$ to $d = 1$ **do**
  $x \leftarrow x + f \cdot \boldsymbol{\alpha}[d]$
  $f \leftarrow f \cdot (p_{max}+1)$
return x

Computing a Multi-index Expansion of a Vector. A multi-index expansion of a vector $\mathbf{x} \in \mathbb{R}^D$ up to p^D terms is basically the set of coefficients $\{\mathbf{x}^{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}<p}$. See Figure 64. This is used in the process of forming a far-field moment contribution of a single reference point in ACCUMULATEFARFIELDMOMENT and evaluating a local expansion in EVALLOCALEXPANSION.

1	$x[2]$	$(x[2])^2$	$(x[2])^3$
$x[1]$	$x[1]x[2]$	$x[1](x[2])^2$	$x[1](x[2])^3$
$(x[1])^2$	$(x[1])^2x[2]$	$(x[1])^2(x[2])^2$	$(x[1])^2(x[2])^3$
$(x[1])^3$	$(x[1])^3x[2]$	$(x[1])^3(x[2])^2$	$(x[1])^3(x[2])^3$

Figure 64: The multi-index expansion of $\mathbf{x} = [\mathbf{x}[1], \mathbf{x}[2]]^T$ up to 16 terms.

Algorithm A.0.3 MULTIINDEXEXPANSION($\mathbf{x}, p, \mathbf{M}'$): Computes $M' = \{\mathbf{x}^{\boldsymbol{\alpha}}\}_{\boldsymbol{\alpha}<p}$.

$\mathbf{M}'[0] \leftarrow 1$
for each $i = 0$ to $i = (p+1)^D - 1$ **do**
  {Retrieve the multi-index mapping of the current position.}
  $\boldsymbol{\alpha} \leftarrow$ POSITIONTOMULTIINDEX(i, p)
  $j \leftarrow$ the first index of $\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha}[j] \geq 1$.
  {Found a direct ancestor of the multiindex map $\boldsymbol{\alpha}$.}
  $\boldsymbol{\alpha}' \leftarrow \boldsymbol{\alpha}, \qquad \boldsymbol{\alpha}'[j] \leftarrow \boldsymbol{\alpha}'[j] - 1$
  {Recursively compute the $\boldsymbol{\alpha}$-th multi-index component based on $\boldsymbol{\alpha}'$-th.}
  $\mathbf{M}'[i] \leftarrow \mathbf{M}'[$MULTIINDEXTOPOSITION$(\boldsymbol{\alpha}')] \cdot \mathbf{x}[j]$

Far-field Moment Accumulation (Equation (3.1.11)**).** This is straightforward given the implementation of the function MULTIINDEXEXPANSION. It computes the multi-index of each reference point in the given reference node and accumulates each

Algorithm A.0.4 AccumulateFarFieldMoment($\mathbf{R}^{sub}$): Equation (3.1.11).

{Temporary space that is equal in size to $\{M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}})\}_{\boldsymbol{\alpha} \leq p_{max}}$.}
$\mathbf{M}'_{i=0,\cdots,(p_{max}+1)^D-1} \leftarrow 0$
for each $\mathbf{r}_{\mathbf{j_n}} \in \mathbf{R}^{sub}$ **do**
  {Add $\mathbf{M}' = \left\{ \left(\frac{\mathbf{r}_{\mathbf{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}} \right\}_{\boldsymbol{\alpha} \leq p_{max}}$ onto $\{M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}})\}_{\boldsymbol{\alpha} \leq p_{max}}$.}
  MultiIndexExpansion $\left(\frac{\mathbf{r}_{\mathbf{j_n}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}, p_{max}, \mathbf{M}' \right)$
  $\{M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}})\}_{\boldsymbol{\alpha} \leq p_{max}} \leftarrow \{M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}})\}_{\boldsymbol{\alpha} \leq p_{max}} + \mathbf{M}'$
for $i = 0$ to $i = (p_{max}+1)^D - 1$ **do**
  $M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \leftarrow M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \cdot \frac{1}{\boldsymbol{\alpha}!}$

contribution and normalizes the sum. See Algorithm A.0.4.

Far-to-Far Translation Operator (shown in Algorithm A.0.5). This consists of a doubly-nested for-loop over accumulated far-field moments.

Algorithm A.0.5 TransFarToFar($\mathbf{R}', \mathbf{R}^{sub}$): Implements Equation (3.1.17).

{Allocate space for and compute $\left\{ \left(\frac{\mathbf{c}_{\mathbf{R}'} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}} \right)^{\boldsymbol{\alpha}} \right\}_{\boldsymbol{\alpha} \leq p_{max}}$.}
$\mathbf{C}_{i=0,\cdots,(p_{max}+1)^D-1} \leftarrow 0$
MultiIndexExpansion $\left(\frac{\mathbf{c}_{\mathbf{R}'} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}, p_{max}, \mathbf{C} \right)$

for $i = 0$ to $i < (p_{max}+1)^D$ **do**
  $\boldsymbol{\gamma} \leftarrow$ PositionToMultiindex(i, p_{max})
  for $j = 0$ to $j < (p_{max}+1)^D$ **do**
    $\boldsymbol{\alpha} \leftarrow$ PositionToMultiindex(j, p_{max})
    if $\boldsymbol{\alpha} \leq \boldsymbol{\gamma}$ **then**
      $M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \leftarrow M_{\boldsymbol{\gamma}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) +$
        $\frac{1}{(\boldsymbol{\gamma}-\boldsymbol{\alpha})!} M_{\boldsymbol{\alpha}}(\mathbf{R}', \mathbf{c}_{\mathbf{R}'}) \quad \cdot \quad \mathbf{C}[\text{MultiIndexToPosition}(\boldsymbol{\gamma} - \boldsymbol{\alpha})]$

Computing the Multivariate Hermite Functions. We exploit the fact that the multivariate Hermite functions is a product of D univariate Hermite functions. Algorithm A.0.6 computes partial derivatives of the Gaussian kernel evaluated at the given point x along each dimension up to p-th order. $h_{\boldsymbol{\alpha}}(x) = \prod_{d=1}^{D} h_{\boldsymbol{\alpha}[d]}(x)$ is a simple product of the univariate functions (see Algorithm A.0.7).

Algorithm A.0.6 ComputePartialDerivatives($\mathbf{a}, p, \mathbf{H}$): Evaluates the partial derivatives of $\exp(-x^2/(2h^2))$ up to p-th order at each coordinate of $\mathbf{a}$.

```
for d = 1 to D do
  H[d][0] ← exp(−(a[d])^2)
  if p > 1 then
    H[d][1] ← 2 · a[d] · exp(−(a[d])^2)
    if p > 2 then
      for k = 1 to k = p − 1 do
        H[d][k + 1] ← 2 · a[d] · H[d][k] − 2 · k · H[d][k − 1]
```

Algorithm A.0.7 ComputeHermiteFunction($\mathbf{H}, \boldsymbol{\alpha}$): Computes the Hermite function $h_{\boldsymbol{\alpha}}(\cdot)$ using the pre-computed partial derivatives $\mathbf{H}$.

```
f ← 1
for d = 1 to D do
  f ← f · H[d][α[d]]
return f
```

Evaluating a Far-field Expansion. Once the functions for computing the Hermite functions (Algorithm A.0.6 and Algorithm A.0.7), we can implement the function for evaluating a far-field expansion up to $\mathcal{O}(p^D)$ terms, as shown in Algorithm A.0.8. The basic structure is one outer-loop over each query point and the inner loop iterating over each far-field moment. The contribution to each query point is computed as a dot-product between the far-field moment and the computed Hermite functions (see Figure 18).

Far-to-Local Translation Operator. The basic structure of the algorithm is a doubly nested for-loop, each over the coefficients. The doubly-nested for-loop first translate a portion of the accumulated far-field moments of $\mathbf{R}^{sub}$ up to p^D terms into the local moments. The final step of the algorithm is to add the translated moments $\widetilde{N}_{\boldsymbol{\beta}}(\{(M(\mathbf{R}^{sub}, c_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})$ to the local moments stored in $\mathbf{Q}^{sub}$, $\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))$. See Algorithm A.0.9.

Algorithm A.0.8 EvalFarFieldExpansion($\mathbf{R}^{sub}, \mathbf{Q}^{sub}, p$): Evaluates the far-field expansion of $\mathbf{R}^{sub}$ up to $(p+1)^D$ terms.

{Allocate space for holding the partial derivatives.}
$\mathbf{H}_{\substack{d=1,\cdots,D \\ k=0,\cdots,p}} \leftarrow 0$

for each $\mathbf{q_{i_m}} \in \mathbf{Q}^{sub}$ **do**
  {Compute partial derivatives up to p-th order along each dimension.}
  ComputePartialDerivatives $\left(\frac{\mathbf{q_{i_m}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}, p, \mathbf{H}\right)$
  $w \leftarrow 0$
  for $i = 0$ to $i = (p+1)^D - 1$ **do**
    $\boldsymbol{\alpha} \leftarrow$ PositionToMultiindex(i, p)
    $f \leftarrow$ ComputeHermiteFunction$(\mathbf{H}, \boldsymbol{\alpha})$
    $w \leftarrow w + M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \cdot f$
  $\widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{DF}}(\mathbf{q_{i_m}})) \leftarrow \widetilde{\Phi}(\mathbf{q_{i_m}}; \mathbf{R}^{\mathsf{DF}}(\mathbf{q_{i_m}})) + w$

Direct Local Accumulation Operation. The basic structure is a doubly-nested for-loop, the outer-loop over the reference points whose moments are to be accumulated as local moments and the inner loop over the coefficient positions. See Algorithm A.0.10.

Local-to-Local Translation Operator. We direct readers' attention to the first step of the algorithm, which retrieves the maximum order among used in local moment accumulation/translation. Then the algorithm proceeds with a doubly-nested for-loop over the local moments applies Equation (3.1.18). See Algorithm A.0.11.

Evaluating the Local Expansion of a Query Node. This function (see Algorithm A.0.12) contains one outer-loop over reference points and the inner-loop over the local moments up to $(p+1)^D$ terms, where p is the maximum approximation order used among the reference nodes pruned via far-to-local and direct local accumulations.

Algorithm A.0.9 TransFarToLocal($\mathbf{R}^{sub}, \mathbf{Q}^{sub}, p$): Implements Equation (3.1.13).

$H_{\substack{d=1,\cdots,D \\ k=0,\cdots,2p}} \leftarrow 0$

ComputePartialDerivatives $\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{c}_{\mathbf{R}^{sub}}}{\sqrt{2h^2}}, 2p, H\right)$

for $i = 0$ to $i = (p+1)^D - 1$ **do**
  $\boldsymbol{\beta} \leftarrow$ PositionToMultiindex(i, p)
  for $j = 0$ to $j = (p+1)^D - 1$ **do**
    $\boldsymbol{\alpha} \leftarrow$ PositionToMultiindex(j, p)
    $f \leftarrow$ ComputeHermiteFunction$(H, \boldsymbol{\alpha} + \boldsymbol{\beta})$
    $\widetilde{N}_{\boldsymbol{\beta}}\left(\left\{M_{\boldsymbol{\beta}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), \mathbf{c}_{\mathbf{Q}^{sub}}, p)\right\}\right) \leftarrow \widetilde{N}_{\boldsymbol{\beta}}\left(\left\{M_{\boldsymbol{\beta}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), \mathbf{c}_{\mathbf{Q}^{sub}}, p)\right\}\right) + M_{\boldsymbol{\alpha}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}) \cdot f$
  $\widetilde{N}_{\boldsymbol{\beta}}\left(\left\{M_{\boldsymbol{\beta}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), \mathbf{c}_{\mathbf{Q}^{sub}}, p)\right\}\right) \leftarrow \frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!} \widetilde{N}_{\boldsymbol{\beta}}\left(\left\{M_{\boldsymbol{\beta}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), \mathbf{c}_{\mathbf{Q}^{sub}}, p)\right\}\right)$
$\{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \le p} \leftarrow \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \le p} + \left\{\widetilde{N}_{\boldsymbol{\beta}}\left(\left\{M_{\boldsymbol{\beta}}(\mathbf{R}^{sub}, \mathbf{c}_{\mathbf{R}^{sub}}), (\mathbf{c}_{\mathbf{Q}^{sub}}, p)\right\}\right)\right\}_{\boldsymbol{\beta} \le p}$

Algorithm A.0.10 AccumulateDirectLocalMoment($\mathbf{R}^{sub}, \mathbf{Q}^{sub}, p$): Implements Equation (3.1.12).

$H_{\substack{d=1,\cdots,D \\ k=0,\cdots,p}} \leftarrow 0, \qquad \{N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta} \le p} \leftarrow 0$
for each $\mathbf{r}_{\mathbf{j_n}} \in \mathbf{R}^{sub}$ **do**
  ComputePartialDerivatives $\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{r}_{\mathbf{j_n}}}{\sqrt{2h^2}}, p, \mathbf{H}\right)$
  for $i = 0$ to $(p+1)^D - 1$ **do**
    $\boldsymbol{\alpha} \leftarrow$ PositionToMultiindex(i, p)
    $f \leftarrow$ ComputeHermiteFunction$(H, \boldsymbol{\beta})$
    $N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta} \le p} \leftarrow N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta} \le p} + f$
$\{N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta} \le p} \leftarrow \{N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta} \le p} \cdot \frac{(-1)^{|\boldsymbol{\beta}|}}{\boldsymbol{\beta}!}$
$\{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \le p} \leftarrow \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \le p} + \{N_{\boldsymbol{\beta}}(\{(\mathbf{R}^{sub}, (\mathbf{c}_{\mathbf{Q}^{sub}}, p))\})\}_{\boldsymbol{\beta} \le p}$

Algorithm A.0.11 TRANSLOCALTOLOCAL($\mathbf{Q}^{sub'}, \mathbf{Q}^{sub}$): Implements Equation (3.1.18).

{p is the maximum approximation order used among the reference nodes pruned via far-to-local and direct local accumulations for $\mathbf{Q}^{sub'}$.}
{Temporary space that is equal in size to $\{\widetilde{N}_{\boldsymbol{\beta}}\}$.}
$\{X\}_{0,\cdots,(p+1)^D-1} \leftarrow 0$
MULTIINDEXEXPANSION $\left(\frac{\mathbf{c}_{\mathbf{Q}^{sub}} - \mathbf{c}_{\mathbf{Q}^{sub'}}}{\sqrt{2h^2}}, p, X\right)$
for $j = 0$ to $(p+1)^D - 1$ **do**
  $\boldsymbol{\alpha} \leftarrow$ POSITIONTOMULTIINDEX(j, p)
  for $k = 0$ to $(p+1)^D - 1$ **do**
    $\boldsymbol{\beta} \leftarrow$ POSITIONTOMULTIINDEX(k, p)
    if $\boldsymbol{\beta} \geq \boldsymbol{\alpha}$ **then**
      $\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub'}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub'})) \leftarrow \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub'}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub'})) +$
        $\frac{\boldsymbol{\beta}!}{\boldsymbol{\alpha}!(\boldsymbol{\beta}-\boldsymbol{\alpha})!} L_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub'}}, \mathcal{R}_{\mathcal{D}}(\mathbf{Q}^{sub'}) \cup \mathcal{R}_{\mathcal{T}}(\mathbf{Q}^{sub'})) X_{\boldsymbol{\beta}-\boldsymbol{\alpha}}$
$\{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \leq p} \leftarrow \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}_{\mathbf{D}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub}))\}_{\boldsymbol{\beta} \leq p} + \{\widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}_{\mathbf{D}}(\mathbf{Q}^{sub'}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub'}))\}_{\boldsymbol{\beta} \leq p}$

Algorithm A.0.12 EVALLOCALEXPANSION($\mathbf{Q}^{sub}$): Evaluates the accumulated local expansion of the given query node $\mathbf{Q}^{sub}$.

{p is the maximum approximation order used among the reference nodes pruned via far-to-local and direct local accumulations for $\mathbf{Q}^{sub}$.}
{Temporary space to hold the multi-index expansion of each $\left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}\right)^{\boldsymbol{\alpha}}$.}
$X_{i=0,\cdots,p^D-1} \leftarrow 0$

for each $\mathbf{q}_{\mathbf{i_m}} \in Q$ **do**
  $z \leftarrow 0$
  {Compute the multi-index expansion of $\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}$ up to p^D terms.}
  MULTIINDEXEXPANSION $\left(\frac{\mathbf{q}_{\mathbf{i_m}} - \mathbf{c}_{\mathbf{Q}^{sub}}}{\sqrt{2h^2}}, p, X\right)$
  for $i = 0$ to $i = p^D - 1$ **do**
    $\boldsymbol{\beta} \leftarrow$ POSITIONTOMULTIINDEX(i, p)
    $z \leftarrow z + \widetilde{N}_{\boldsymbol{\beta}}(\mathbf{c}_{\mathbf{Q}^{sub}}, \mathbf{R}^{\mathsf{DL}}(\mathbf{Q}^{sub}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{Q}^{sub})) \cdot z$
  $\widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}^{\mathsf{DL}}(\mathbf{q}_{\mathbf{i_m}}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{q}_{\mathbf{i_m}})) \leftarrow \widetilde{\Phi}(\mathbf{q}_{\mathbf{i_m}}; \mathbf{R}^{\mathsf{DL}}(\mathbf{q}_{\mathbf{i_m}}) \cup \mathbf{R}^{\mathsf{F2L}}(\mathbf{q}_{\mathbf{i_m}})) + z$

REFERENCES

[1] "NERSC computational systems." `http://www.nersc.gov/users/computational-systems/`.

[2] Achlioptas, D., McSherry, F., and Scholkopf, B., "Sampling techniques for kernel methods," *Advances in Neural Information Processing Systems*, vol. 1, pp. 335–342, 2002.

[3] Al-Furajh, I., Aluru, S., Goil, S., and Ranka, S., "Parallel construction of multidimensional binary search trees," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 11, no. 2, pp. 136–148, 2002.

[4] Aluru, S., Prabhu, G., and Gustafson, J., "Truly distribution-independent algorithms for the N-body problem," in *Proceedings of the 1994 conference on Supercomputing table of contents*, pp. 420–428, IEEE Computer Society Press Los Alamitos, CA, USA, 1994.

[5] Anderson, C., "An implementation of the fast multipole method without multipoles," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, p. 923, 1992.

[6] Anjyo, K. and Lewis, J., "Rbf interpolation and gaussian process regression through an rkhs formulation," *Journal of Math-for-Industry*, vol. 3, no. A, pp. 63–71, 2011.

[7] Appel, A., "An efficient program for many-body simulation," *SIAM Journal on Scientific and Statistical Computing*, vol. 6, p. 85, 1985.

[8] Auffray, Y., Barbillon, P., and others, "Conditionally positive definite kernels: theoretical contribution, application to interpolation and approximation," 2009.

[9] Axilrod, B. and Teller, E., "Interaction of the van der Waals type between three atoms," *The Journal of Chemical Physics*, vol. 11, p. 299, 1943.

[10] Barnes, J. and Hut, P., "A Hierarchical $O(NlogN)$ Force-Calculation Algorithm," *Nature*, vol. 324, 1986.

[11] Baxter, B. and Roussos, G., "A new error estimate of the fast Gauss transform," *SIAM Journal on Scientific Computing*, vol. 24, p. 257, 2002.

[12] Beatson, R. and Greengard, L., "A short course on fast multipole methods," *Wavelets, Multilevel Methods and Elliptic PDEs*, pp. 1–37, 1997.

[13] Belkin, M. and Niyogi, P., "Laplacian eigenmaps and spectral techniques for embedding and clustering," *Advances in neural information processing systems*, vol. 14, pp. 585–591, 2001.

[14] Bengio, Y., Vincent, P., Paiement, J., Delalleau, O., Ouimet, M., and LeRoux, N., "Learning eigenfunctions of similarity: Linking spectral clustering and kernel pca," *Dpartement dinformatique et recherche oprationnelle, Universit de Montral, Tech. Rep*, 2003.

[15] Bentley, J. L., "Multidimensional Binary Search Trees used for Associative Searching," *Communications of the ACM*, vol. 18, pp. 509–517, 1975.

[16] Bentley, J., "Multidimensional divide-and-conquer," *Communications of the ACM*, vol. 23, no. 4, pp. 214–229, 1980.

[17] Bentley, J., "K-d trees for semidynamic point sets," in *Proceedings of the sixth annual symposium on Computational geometry*, pp. 187–197, ACM, 1990.

[18] Benzi, M., Golub, G., and Liesen, J., "Numerical solution of saddle point problems," *Acta numerica*, vol. 14, no. 1, pp. 1–137, 2005.

[19] Bertsekas, D. P. and Tsitsiklis, J. N., *Parallel and Distributed Computation*. Prentice Hall, 1989.

[20] Beygelzimer, A., Kakade, S., and Langford, J., "Cover trees for nearest neighbor," in *Proceedings of the 23rd international conference on Machine learning*, pp. 97–104, ACM New York, NY, USA, 2006.

[21] Bialecki, A., Cafarella, M., Cutting, D., and OMALLEY, O., "Hadoop: a framework for running applications on large clusters built of commodity hardware," *Wiki at http://lucene. apache. org/hadoop*, vol. 11, 2005.

[22] Bo, L. and Sminchisescu, C., "Greedy block coordinate descent for large scale gaussian process regression," in *Uncertainty in Artificial Intelligence*, Citeseer, 2008.

[23] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J., *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers, 2011.

[24] Boyer, G., Riegel, R., Vasiloglou, N., Lee, D., Poorman, L., Mappus, C., Mehta, N., Ouyang, H., Ram, P., Tran, L., Wong, W. C., and Gray, A., "MLPACK." `http://mloss.org/software/view/152`, 2009.

[25] Boyer, G., Riegel, R., and Gray, A., "A parallel n-body data mining framework," in *NIPS Workshop on Efficient Machine Learning*, 2007.

[26] Brinkhoff, T., Kriegel, H., and Seeger, B., "Parallel processing of spatial joins using r-trees," in *Data Engineering, 1996. Proceedings of the Twelfth International Conference on*, pp. 258–265, IEEE, 1996.

[27] Burtscher, M. and Pingali, K., "An efficient cuda implementation of the tree-based barnes hut n-body algorithm," *GPU Computing Gems Emerald Edition*, p. 75, 2011.

[28] Callahan, P. B., *Dealing with Higher Dimensions: The Well-Separated Pair Decomposition and its Applications*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, 1995.

[29] Callahan, P., *Dealing with higher dimensions: the well-separated pair decomposition and its applications*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, 1995.

[30] Callahan, P. and Kosaraju, S., "A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields," *Journal of the ACM (JACM)*, vol. 42, no. 1, pp. 67–90, 1995.

[31] Cayton, L., "A nearest neighbor data structure for graphics hardware," in *Proceedings of International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*, 2010.

[32] Cayton, L., "Accelerating nearest neighbor search on manycore systems," *Arxiv preprint arXiv:1103.2635*, 2011.

[33] Chavez, E., Navarro, G., Baeza-Yates, R., and Marroqun, J. L., "Proximity searching in metric spaces," *ACM Computing Surveys*, vol. 33, no. 3, pp. 273–321, 2001.

[34] Chen, A., "Fast kernel density independent component analysis," *Independent Component Analysis and Blind Signal Separation*, pp. 24–31, 2006.

[35] Chen, G. and Lerman, G., "Spectral curvature clustering (scc)," *International journal of computer vision*, vol. 81, no. 3, pp. 317–330, 2009.

[36] Cheng, H., Gimbutas, Z., Martinsson, P., and Rokhlin, V., "On the compression of low rank matrices," *SIAM Journal on Scientific Computing*, vol. 26, no. 4, pp. 1389–1404, 2005.

[37] Cheng, H., Greengard, L., and Rokhlin, V., "A fast adaptive multipole algorithm in three dimensions," *Journal of Computational Physics*, vol. 155, no. 2, pp. 468–498, 1999.

[38] Cheng, Y., "Mean shift, mode seeking, and clustering," *IEEE Trans. Pattern Anal. Mach. Intel.*, vol. 17, no. 8, pp. 790–799, 1995.

[39] Choi, B., Komuravelli, R., Lu, V., Sung, H., Bocchino, R., Adve, S., and Hart, J., "Parallel sah kd tree construction," in *Proceedings of the Conference on High Performance Graphics*, pp. 77–86, Eurographics Association, 2010.

[40] Chu, C., Kim, S., Lin, Y., Yu, Y., Bradski, G., Ng, A., and Olukotun, K., "Map-reduce for machine learning on multicore," in *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, p. 281, The MIT Press, 2007.

[41] Clarkson, K., Hazan, E., and Woodruff, D., "Sublinear optimization for machine learning," *Arxiv preprint arXiv:1010.4408*, 2010.

[42] Clarkson, K. and Woodruff, D., "Numerical linear algebra in the streaming model," in *Proceedings of the 41st annual ACM symposium on Theory of computing*, pp. 205–214, ACM, 2009.

[43] Cleveland, W. S., "Robust Locally Weighted Regression and Smoothing Scatterplots," *Journal of the American Statistical Association*, vol. 74, pp. 829–836, December 1979.

[44] Cleveland, W. S. and Delvin, S. J., "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting," *Journal of the American Statistical Association*, vol. 83, pp. 596–610, September 1988.

[45] Cleveland, W. S. and Grosse, E., "Computational Methods for Local Regression," *Statistics and Computing*, vol. 1, no. 1, pp. 47–62, 1991.

[46] Collins, R., "Mean-shift blob tracking through scale space," in *Conf. on Computer Vision and Pattern Rec.*, vol. 2, pp. 234–240, 2003.

[47] Comaniciu, D., Ramesh, V., and Meer, P., "Real-time tracking of non-rigid objects using mean shift," in *Conf. on Computer Vision and Pattern Rec.*, pp. 142 – 149, June 2000.

[48] Comaniciu, D. and Meer, P., "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intel.*, vol. 24, no. 5, pp. 603–619, 2002.

[49] Contributors, G. P., "GSL - GNU scientific library - GNU project - free software foundation (FSF)." http://www.gnu.org/software/gsl/, 2010.

[50] Cortes, C., Mohri, M., and Talwalkar, A., "On the impact of kernel approximation on learning accuracy," in *Conference on Artificial Intelligence and Statistics*, pp. 113–120, 2010.

[51] Coulaud, O., Fortin, P., and Roman, J., "High-performance BLAS formulation of the adaptive fast multipole method," *Advances in Computational Methods in Sciences and Engineering*, pp. 1796–1799, 2005.

[52] Cruz, F., Knepley, M., and Barba, L., "PetFMM–A dynamically load-balancing parallel fast multipole library," *Arxiv preprint arXiv:0905.2637*, 2009.

[53] Curtin, R., Cline, J., Slagle, N., Amidon, M., Kale, A., March, B., Mehta, N., Ram, P., Lee, D., and Gray, A., "libmlpack," 2011. `http://mloss.org/software/view/364/`.

[54] Dasgupta, S. and Freund, Y., "Random projection trees and low dimensional manifolds," in *Proceedings of the 40th annual ACM symposium on Theory of computing*, pp. 537–546, ACM, 2008.

[55] Dasgupta, S. and Freund, Y., "Random projection trees and low dimensional manifolds," tech. rep., 2008.

[56] de Freitas, N., Wang, Y., Mahdaviani, M., and Lang, D., "Fast krylov methods for n-body learning," in *Advances in Neural Information Processing Systems 18* (Weiss, Y., Schölkopf, B., and Platt, J., eds.), pp. 251–258, Cambridge, MA: MIT Press, 2006.

[57] Delaigle, A. and Meister, A., "Nonparametric regression estimation in the heteroscedastic errors-in-variables problem," *Journal of the American Statistical Association*, vol. 102, no. 480, pp. 1416–1426, 2007.

[58] Deng, K. and Moore, A. W., "Multiresolution Instance-based Learning," in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, (San Francisco), pp. 1233–1239, Morgan Kaufmann, 1995.

[59] Dhesi, A. and Kar, P., "Random projection trees revisited," *Arxiv preprint arXiv:1010.3812*, 2010.

[60] Drineas, P., Kannan, R., and Mahoney, M., "Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix," *SIAM Journal on Computing*, vol. 36, no. 1, p. 158, 2006.

[61] Drineas, P. and Mahoney, M., "On the Nystrom method for approximating a Gram matrix for improved kernel-based learning," *The Journal of Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[62] El Karoui, N. and dAspremont, A., "Second order accurate distributed eigenvector computation for extremely large matrices," *Electronic Journal of Statistics*, vol. 4, pp. 1345–1385, 2010.

[63] Faloutsos, C. and Lin, K., *FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets*, vol. 24. ACM, 1995.

[64] Faloutsos, C., Seeger, B., Traina, A., and Traina Jr, C., *Spatial join selectivity using power laws*, vol. 29. ACM, 2000.

[65] Fashing, M. and Tomasi, C., "Mean shift is a bound optimization," *IEEE Trans. Pattern Anal. Mach. Intel.*, vol. 27, no. 3, pp. 471–474, 2005.

[66] Fine, S. and Scheinberg, K., "Efficient SVM training using low-rank kernel representations," *The Journal of Machine Learning Research*, vol. 2, pp. 243–264, 2002.

[67] Fox, G., "Numerical algorithms for modern parallel computer architectures, chapter a graphical approach to load balancing and sparse matrix vector multiplication on the hypercube," 1988.

[68] Frank, A. and Asuncion, A., "UCI machine learning repository," 2010.

[69] Franklin, M. and Govindan, V., "The n-body problem: Distributed system load balancing and performance evaluation," in *In Proceedings of the 6th International Conference on Parallel and Distributed Computing Systems*, Citeseer, 1993.

[70] Friedman, J. H., Bentley, J. L., and Finkel, R. A., "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[71] Fukunaga, K. and Hostetler, L. D., "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Trans. on Information Theory*, vol. 21, no. 1, pp. 32–40, 1975.

[72] Gardner, J., Connolly, A., and McBride, C., "A framework for analyzing massive astrophysical datasets on a distributed grid," in *ASTRONOMICAL SOCIETY OF THE PACIFIC CONFERENCE SERIES*, vol. 376, p. 69, ASP; 1999, 2007.

[73] Georgescu, B., Shimshoni, I., and Meer, P., "Mean shift based clustering in high dimensions: A texture classification example," in *Intl. Conf. on Computer Vision*, pp. 456–463, 2003.

[74] Gionis, A., Indyk, P., and Motwani, R., "Similarity search in high dimensions via hashing," in *VLDB*, pp. 518–529, 1999.

[75] Glasserman, P., *Monte Carlo methods in financial engineering*. Springer Verlag, 2004.

[76] Golub, G., *Matrix Computations, Third Edition*. The Johns Hopkins University Press, 1996.

[77] Gray, A. and Moore, A., "Rapid evaluation of multiple density models," 2003.

[78] Gray, A. and Moore, A. W., "N-Body Problems in Statistical Learning," in *Advances in Neural Information Processing Systems 13 (December 2000)* (Leen, T. K., Dietterich, T. G., and Tresp, V., eds.), MIT Press, 2001.

[79] Gray, A. and Riegel, R., "Large-scale kernel discriminant analysis with application to quasar discovery," *Proceedings of Computational Statistics*, 2006.

[80] Gray, A. and Moore, A., "Very fast multivariate kernel density estimation via computational geometry," in *Joint Stat. Meeting*, 2003.

[81] Gray, A. G., Lee, D., Rotella, C., and Moore, A. W., "What is the best nearest-neighbor method?." 2004.

[82] Gray, A. G. and Moore, A. W., "Nonparametric Density Estimation: Toward Computational Tractability," in *SIAM International Conference on Data Mining*, 2003.

[83] Greengard, L. and Rokhlin, V., "A Fast Algorithm for Particle Simulations," *Journal of Computational Physics*, vol. 73, 1987.

[84] Greengard, L. and Strain, J., "The Fast Gauss Transform," *SIAM Journal of Scientific and Statistical Computing*, vol. 12(1), pp. 79–94, 1991.

[85] Greengard, L. and Sun, X., "A new version of the fast Gauss transform," *Documenta Mathematica*, vol. 901, no. 3, pp. 575–584, 1998.

[86] Greengard, L. and Huang, J., "A new version of the fast multipole method for screened Coulomb interactions in three dimensions," *Journal of Computational Physics*, vol. 180, no. 2, pp. 642–658, 2002.

[87] Grosse, E., "LOESS: Multivariate smoothing by moving least squares," *Approximation Theory VI*, vol. 2, pp. 373–376, 1989.

[88] Gunter, S., Schraudolph, N., and Vishwanathan, S., "Fast iterative kernel principal component analysis," *Journal of Machine Learning Research*, vol. 8, pp. 1893–1918, 2007.

[89] Halko, N., Martinsson, P., and Tropp, J., "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM review*, vol. 53, no. 2, pp. 217–288, 2011.

[90] Hall, P., Marshall, D., and Martin, R., "Merging and splitting eigenspace models," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 9, pp. 1042–1049, 2000.

[91] Ham, J., Lee, D., Mika, S., and Schölkopf, B., "A kernel view of the dimensionality reduction of manifolds," in *Proceedings of the twenty-first international conference on Machine learning*, p. 47, ACM, 2004.

[92] Hammersley, J., "The distribution of distance in a hypersphere," *The Annals of Mathematical Statistics*, vol. 21, no. 3, pp. 447–452, 1950.

[93] Heroux, M., Bartlett, R., Howle, V., Hoekstra, R., Hu, J., Kolda, T., Lehoucq, R., Long, K., Pawlowski, R., Phipps, E., and others, "An overview of the trilinos project," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 397–423, 2005.

[94] Holmes, M., *Multi-tree Monte Carlo methods for fast, scalable machine learning*. PhD thesis, Georgia Institute of Technology, 2009.

[95] Holmes, M., Gray, A., and Isbell Jr, C., "Ultrafast Monte Carlo for kernel estimators and generalized statistical summations," *Advances in Neural Information Processing Systems (NIPS)*, vol. 21, 2008.

[96] Holmes, M., Gray, A., and Isbell Jr, C., "Fast kernel conditional density estimation: A dual-tree Monte Carlo approach," *Computational Statistics & Data Analysis*, vol. 54, no. 7, pp. 1707–1718, 2010.

[97] Hu, Q., Gumerov, N., and Duraiswami, R., "Scalable fast multipole methods on distributed heterogeneous architectures," in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pp. 1–12, IEEE, 2011.

[98] Kang, U., Tsourakakis, C., and Faloutsos, C., "Pegasus: A peta-scale graph mining system implementation and observations," in *Data Mining, 2009. ICDM'09. Ninth IEEE International Conference on*, pp. 229–238, IEEE, 2009.

[99] Kapur, S. and Long, D., "IES 3: efficient electrostatic and electromagnetic-simulation," *IEEE Computational Science & Engineering*, vol. 5, no. 4, pp. 60–67, 1998.

[100] Kar, P. and Karnick, H., "Random feature maps for dot product kernels," in *In Artificial Intelligence and Statistics (AISTATS)*, 2012.

[101] Keerthi, S. and Chu, W., "A matching pursuit approach to sparse gaussian process regression," *Advances in neural information processing systems*, vol. 18, p. 643, 2006.

[102] Khan, U., Kar, S., and Moura, J., "Distributed average consensus: Beyond the realm of linearity," in *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pp. 1337–1342, IEEE, 2009.

[103] Kim, K., Lee, D., and Essa, I., "Gaussian process regression flow for analysis of motion trajectories," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, IEEE Computer Society, November 2011.

[104] Kim, K., Lee, D., and Essa, I., "Detecting regions of interest in dynamic scenes with camera motions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012*, IEEE Computer Society, 2012.

[105] Klaas, M., Lang, D., and de Freitas, N., "Fast maximum a posteriori inference in monte carlo state spaces," in *Artificial Intelligence and Statistics*, 2005.

[106] Kobayashi, K., "A remark on the fast gauss transform," *Publications of the Research Institute for Mathematical Sciences*, vol. 39, no. 4, pp. 785–796, 2003.

[107] Koranne, S., "Boost c++ libraries," *Handbook of Open Source Tools*, pp. 127–143, 2011.

[108] Kruskal, J., "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, no. 1, pp. 1–27, 1964.

[109] Kubica, J. and Moore, A., "Efficient discovery of spatial associations and structure with application to asteroid tracking," 2005.

[110] Lashuk, I., Chandramowlishwaran, A., Langston, H., Nguyen, T., Sampath, R., Shringarpure, A., Vuduc, R., Ying, L., Zorin, D., and Biros, G., "A massively parallel adaptive fast-multipole method on heterogeneous architectures," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pp. 1–12, ACM, 2009.

[111] LeCun, Y., "MNIST dataset," 2000. http://yann.lecun.com/exdb/mnist/.

[112] Lee, D. and Gray, A., "Faster gaussian summation: Theory and experiment," in *Proceedings of the Twenty-second Conference on Uncertainty in Artificial Intelligence*, 2006.

[113] Lee, D. and Gray, A., "Fast high-dimensional kernel summations using the monte carlo multipole method," in *In Advances in Neural Information Processing Systems 21*, 2009.

[114] Lee, D., Gray, A., and Moore, A., "Dual-tree fast gauss transforms," in *Advances in Neural Information Processing Systems 18* (Weiss, Y., Schölkopf, B., and Platt, J., eds.), pp. 747–754, Cambridge, MA: MIT Press, 2006.

[115] Lee, D., Gray, A. G., and Moore, A. W., "Dual-tree fast gauss transforms," 2011.

[116] Lee, D., Ozakin, A., and Gray, A. G., "Multibody multipole methods," *Under review in Journal of Computational Physics.*, 2012.

[117] Lee, D., Vuduc, R. W., and Gray, A. G., "A distributed kernel summation framework for general-dimension machine learning," *SIAM International Conference on Data Mining*, 2012.

[118] Li, J., Zhou, Z., and Sadus, R., "Modified force decomposition algorithms for calculating three-body interactions via molecular dynamics," *Computer Physics Communications*, vol. 175, no. 11-12, pp. 683–691, 2006.

[119] Liu, P. and jan Wu, J., "A framework for parallel tree-based scientific simulations," in *Proceedings of 26 th International Conference on Parallel Processing*, pp. 137–144, 1997.

[120] Liu, T., Rosenberg, C., and Rowley, H., "Clustering Billions of Images with Large Scale Nearest Neighbor Search," in *Proceedings of the Eighth IEEE Workshop on Applications of Computer Vision*, p. 28, IEEE Computer Society, 2007.

[121] Liu, T., Moore, A. W., and Gray, A., "Efficient exact k-nn and nonparametric classification in high dimensions," in *Advances in Neural Information Processing Systems 16* (Thrun, S., Saul, L., and Schölkopf, B., eds.), Cambridge, MA: MIT Press, 2004.

[122] Loebman, S., Nunley, D., Kwon, Y., Howe, B., Balazinska, M., and Gardner, J., "Analyzing massive astrophysical datasets: Can pig/hadoop or a relational dbms help?," in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pp. 1–10, IEEE, 2009.

[123] Loh, P., Hsu, W., Wentong, C., and Sriskanthan, N., "How network topology affects dynamic loading balancing," *Parallel & Distributed Technology: Systems & Applications, IEEE*, vol. 4, no. 3, pp. 25–35, 1996.

[124] London, K., Moore, S., Mucci, P., Seymour, K., and Luczak, R., "The PAPI cross-platform interface to hardware performance counters," in *Department of Defense Users Group Conference Proceedings*, Citeseer, 2001.

[125] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J., "Graphlab: A new parallel framework for machine learning," in *Conference on Uncertainty in Artificial Intelligence*, pp. 1807–1827, 2010.

[126] Marcelli, G., Todd, B., and Sadus, R., "Beyond traditional effective intermolecular potentials and pairwise interactions in molecular simulation," *Computational ScienceICCS 2002*, pp. 932–941, 2009.

[127] Marcelli, G., "The Role of Three-Body Interactions on the Equilibrium and Non-Equilibrium Properties of Fluids from Molecular Simulation," PhD. Thesis, Swinburne University of Technology, Department of Computer Science, 2001.

[128] March, W. B., Ozakin, A., Lee, D., Riegel, R., and Gray, A. G., *Multi-Tree Algorithms for Large-Scale Astrostatistics*, ch. 21. CRC Press, 2012.

[129] Martinsson, P., Rokhlin, V., and of Computer Science, Y. U. D., "An accelerated kernel-independent fast multipole method in one dimension," *SIAM Journal on Scientific Computing*, vol. 29, no. 3, p. 1160, 2008.

[130] Melkumyan, A. and Ramos, F., "A sparse covariance function for exact gaussian process inference in large datasets," *The 21st IJCAI*, 2009.

[131] Mooij, J., "libdai: A free and open source c++ library for discrete approximate inference in graphical models," *The Journal of Machine Learning Research*, vol. 99, pp. 2169–2173, 2010.

[132] Moore, A., Connolly, A., Genovese, C., Gray, A., Grone, L., Kanidoris, N., Nichol, R., Schneider, J., Szalay, A., Szapudi, I., and Wasserman, L., "Fast algorithms and efficient statistics: N-point correlation functions," in *Proceedings of MPA/MPE/ESO Conference Mining the Sky, July 31–August 4, Garching, Germany*, 2000.

[133] Moore, A. and Lee, M., "Cached sufficient statistics for efficient machine learning with large datasets," *Arxiv preprint cs/9803102*, 1998.

[134] Moore, A. W., "The Anchors Hierarchy: Using the Triangle Inequality to Survive High-Dimensional Data," in *Twelfth Conference on Uncertainty in Artificial Intelligence*, AAAI Press, 2000.

[135] Moore, A., "An intoductory tutorial on kd-trees," *Extract from Andrew Moore's PhD Thesis: Effcient Memory based Learning for Robot Control*, 1991.

[136] Moore, A., Connolly, A., Genovese, C., Gray, A., Grone, L., II, N., Nichol, R., Schneider, J., Szalay, A., Szapudi, I., and others, "Fast algorithms and efficient statistics: N-point correlation functions," in *Proceedings of the MPA/MPE/ESO Conference on Mining the Sky*, Springer, 2000.

[137] Moore, A., Schneider, J., and Deng, K., "Efficient locally weighted polynomial regression predictions," in *Proceedings of the 1997 International Machine Learning Conference. Morgan Kaufmann*, Citeseer, 1997.

[138] Nadaraya, E., "On estimating regression," *Theory of Prob. and Appl.*, vol. 9, pp. 141–142, 1964.

[139] Nocedal, J. and Wright, S., "Numerical Optimization, Series in Operations Research and Financial Engineering," 2006.

[140] Omohundro, S. M., "Five Balltree Construction Algorithms," Technical Report TR-89-063, International Computer Science Institute, 1989.

[141] Ouimet, M. and Bengio, Y., "Greedy spectral embedding," in *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, pp. 253–260, Citeseer, 2005.

[142] Ozakin, A. and Gray, A., "Submanifold density estimation," *Advances in Neural Information Processing Systems*, vol. 22, 2009.

[143] Pacheco, P., *Parallel programming with MPI*. Morgan Kaufmann, 1997.

[144] Park, C. and Park, H., "Nonlinear feature extraction based on centroids and kernel functions," *Pattern Recognition*, vol. 37, no. 4, pp. 801–810, 2004.

[145] Parzen, E., "On estimation of a probability density function and mode," *The annals of mathematical statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[146] Pavlo, A., Paulson, E., Rasin, A., Abadi, D., DeWitt, D., Madden, S., and Stonebraker, M., "A comparison of approaches to large-scale data analysis," in *Proceedings of the 35th SIGMOD international conference on Management of data*, pp. 165–178, ACM, 2009.

[147] Pelleg, D. and Moore, A. W., "Accelerating Exact k-means Algorithms with Geometric Reasoning," in *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining*, ACM, 1999.

[148] Pelleg, D. and Moore, A. W., "X-means: Extending K-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning*, (San Francisco), Morgan Kaufmann, 2000.

[149] Plimpton, S., "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995.

[150] Rahimi, A. and Recht, B., "Random features for large-scale kernel machines," *Advances in neural information processing systems*, vol. 20, pp. 1177–1184, 2008.

[151] Ram, P., Lee, D., March, W., and Gray, A., "Linear time algorithms for pairwise statistical problems," *Advances in Neural Information Processing Systems*, vol. 23, 2009.

[152] Ram, P., Lee, D., Ouyang, H., and Gray, A., "Rank-approximate nearest neighbor search: Retaining meaning and speed in high dimensions," *Advances in Neural Information Processing Systems*, vol. 23, 2009.

[153] Rasmussen, C. E. and Williams, C. K. I., *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[154] Raykar, V. C., Yang, C., Duraiswami, R., and Gumerov, N., "Fast computation of sums of gaussians in high dimensions," Tech. Rep. CS-TR-4767, Department of Computer Science, University of Maryland, College Park, 2005.

[155] Ridgway, R., Irfanoglu, O., Machiraju, R., and Huang, K., "Image segmentation with tensor-based classification of n-point correlation functions," in *MICCAI Workshop on Medical Image Analysis with Applications in Biology*, pp. 300–303, 2006.

[156] Riegel, R., Gray, A., and Richards, G., "Massive-scale kernel discriminant analysis: Mining for quasars," in *SIAM International Conference on Data Mining*, Citeseer, 2008.

[157] Roweis, S. and Saul, L., "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[158] Rudin, W., *Fourier analysis on groups*. Wiley-Interscience, 1990.

[159] Saad, Y., *Iterative methods for sparse linear systems*. Society for Industrial Mathematics, 2003.

[160] Salmon, J. K., *Parallel Hierarchical N-body Methods*. PhD thesis, Ph. D. Thesis, California Institute of Technology, 1990.

[161] Samet, H., *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.

[162] Sampath, R., Sundar, H., and Veerapaneni, S., "Parallel Fast Gauss Transform," in *Supercomputing*, 2010.

[163] Sanderson, C., "Armadillo: An open source c++ linear algebra library for fast prototyping and computationally intensive experiments," tech. rep., NICTA, Australia, 2010.

[164] Scholkopf, B., Smola, A., and Muller, K., "Nonlinear component analysis as a kernel eigenvalue problem," *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[165] Scholkopf, B. and Smola, A., "Learning with Kernels: Support Vector Machines, Regularization," *Optimization, and Beyond. MIT Press*, vol. 1, p. 2, 2002.

[166] Scott, D., *Multivariate density estimation: theory, practice, and visualization*. Wiley-Interscience, 1992.

[167] Seeger, M., Williams, C., Lawrence, N., and Dp, S., "Fast forward selection to speed up sparse gaussian process regression," in *in Workshop on AI and Statistics 9*, 2003.

[168] Shende, S. and Malony, A., "The TAU parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, p. 287, 2006.

[169] Shi, Q., Petterson, J., Dror, G., Langford, J., Smola, A., and Vishwanathan, S., "Hash kernels for structured data," *The Journal of Machine Learning Research*, vol. 10, pp. 2615–2637, 2009.

[170] Silverman, B. W., *Density Estimation for Statistics and Data Analysis*. Chapman and Hall/CRC, 1986.

[171] Silverman, B., "Kernel Density Estimation using the Fast Fourier Transform," *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 33, 1982.

[172] Singh, J., Hennessy, J., and Gupta, A., "Implications of hierarchical n-body methods for multiprocessor architectures," *ACM Transactions on Computer Systems (TOCS)*, vol. 13, no. 2, pp. 141–202, 1995.

[173] Singh, J., Holt, C., Totsuka, T., Gupta, A., and Hennessy, J., "Load balancing and data locality in adaptive hierarchical n-body methods: Barnes-hut, fast multipole, and radiosity," *Journal of Parallel and Distributed Computing*, vol. 27, no. 2, pp. 118–141, 1995.

[174] Skeel, R., Tezcan, I., and Hardy, D., "Multiple grid methods for classical molecular dynamics," *Journal of Computational Chemistry*, vol. 23, no. 6, pp. 673–684, 2002.

[175] Smola, A. and Bartlett, P., "Sparse greedy Gaussian process regression," *Advances in Neural Information Processing Systems 13*, 2001.

[176] Smola, A. and Scholkopf, B., "Sparse greedy matrix approximation for machine learning," *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 911–918, 2000.

[177] Song, L., Gretton, A., Bickson, D., Low, Y., and Guestrin, C., "Kernel belief propagation," in *In Artificial Intelligence and Statistics (AISTATS)*, (Ft. Lauderdale, FL), May 2011.

[178] Song, Y., Chen, W., Bai, H., Lin, C., and Chang, E., "Parallel spectral clustering," *Machine Learning and Knowledge Discovery in Databases*, pp. 374–389, 2008.

[179] Spivak, M., Veerapaneni, S. K., and Greengard, L., "The Fast Generalized Gauss Transform," *SIAM Journal of Scientific Computing*, vol. 32, no. 5, pp. 3092–3107, 2010.

[180] Sproull, R. F., "Refinements to Nearest-neighbor Searching," *Algorithmica*, vol. 6, pp. 579–589, 1991.

[181] Srinivasan, B., Hu, Q., and Duraiswami, R., "Gpuml: Graphical processors for speeding up kernel machines," in *Workshop on High Performance Analytics-Algorithms, Implementations, and Applications*, Citeseer, 2010.

[182] Strain, J., "Fast adaptive methods for the free-space heat equation," *SIAM Journal on Scientific Computing*, vol. 15, p. 185, 1994.

[183] Strain, J., "The fast Gauss transform with variable scales," *SIAM Journal on Scientific and Statistical Computing*, vol. 12, no. 5, pp. 1131–1139, 1991.

[184] Szász, O., "On the relative extrema of the hermite orthogonal functions," *J. Indian Math. Soc.*, vol. 15, pp. 129–134, 1951.

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

[186] Tenenbaum, J., De Silva, V., and Langford, J., "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[187] Thiesson, B. and Kim, J., "Fast variational mode-seeking," in *Artificial Intelligence and Statistics*, 2012.

[188] Tron, R. and Vidal, R., "Distributed computer vision algorithms through distributed averaging," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 57 –63, june 2011.

[189] Varagnolo, D., Pillonetto, G., and Schenato, L., "Distributed parametric and nonparametric regression with on-line performance bounds computation," *Automatica*, 2011.

[190] Vasiloglou, N., *Isometry and Convexity in Dimensionality Reduction*. PhD thesis, Georgia Institute of Technology, 2009.

[191] Verma, N., Kpotufe, S., and Dasgupta, S., "Which spatial partition trees are adaptive to intrinsic dimension?," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 565–574, AUAI Press, 2009.

[192] Wand, M. P., "Fast Computation of Multivariate Kernel Estimators," *Journal of Computational and Graphical Statistics*, 1994.

[193] Wang, P., Lee, D., Gray, A., and Rehg, J. M., "Fast mean shift with accurate and stable convergence," in *Artificial Intelligence and Statistics 2007* (Melia, M. and Shen, X., eds.), 2007.

[194] Warren, M. and Salmon, J., "A portable parallel particle program," *Computer Physics Communications*, vol. 87, no. 1, pp. 266–290, 1995.

[195] Williams, C. and Seeger, M., "Using the Nystrom method to speed up kernel machines," in *Advances in Neural Information Processing Systems 13*, Citeseer, 2001.

[196] Williams, C., "On a connection between kernel pca and metric multidimensional scaling," *Machine Learning*, vol. 46, no. 1, pp. 11–19, 2002.

[197] Wissel, D., "Die diskrete gauß-transformation–schnelle approximationsverfahren und anwendungen in hohen dimensionen," 2008.

[198] Wu, G., Zhang, Z., and Chang, E., "Kronecker factorization for speeding up kernel machines," in *SIAM Int. Conference on Data Mining (SDM)*, Citeseer, 2005.

[199] Xiao, L. and Boyd, S., "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004.

[200] Yamamoto, Y., "Efficient parallel implementation of a weather derivatives pricing algorithm based on the fast gauss transform," in *20th International Parallel and Distributed Processing Symposium. 2006, p. 8*, 2006.

[201] Yang, C., Duraiswami, R., Gumerov, N. A., and Davis, L., "Improved fast gauss transform and efficient kernel density estimation," *International Conference on Computer Vision*, 2003.

[202] Yianilos, P., "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pp. 311–321, Society for Industrial and Applied Mathematics, 1993.

[203] Ying, L., Biros, G., and Zorin, D., "A kernel-independent adaptive fast multipole algorithm in two and three dimensions," *Journal of Computational Physics*, vol. 196, no. 2, pp. 591–626, 2004.

[204] Yokota, R., Barba, L., and Knepley, M., "PetRBF–A parallel O (N) algorithm for radial basis function interpolation with Gaussians," *Computer Methods in Applied Mechanics and Engineering*, vol. 199, no. 25-28, pp. 1793–1804, 2010.

[205] Yoo, A., Chow, E., Henderson, K., McLendon, W., Hendrickson, B., and Catalyurek, U., "A scalable distributed parallel breadth-first search algorithm on bluegene/l," in *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, pp. 25–25, IEEE, 2005.

[206] York, D., Adelman, J., Anderson Jr, J., Anderson, S., Annis, J., Bahcall, N., Bakken, J., Barkhouser, R., Bastian, S., Berman, E., and others, "The sloan digital sky survey: Technical summary," *The Astronomical Journal*, vol. 120, p. 1579, 2000.

[207] Zhang, K. and Kwok, J., "Block-quantized kernel matrix for fast spectral embedding," in *Proceedings of the 23rd international conference on Machine learning*, pp. 1097–1104, ACM, 2006.

[208] Zhang, Z. and Zha, H., "Principal manifolds and nonlinear dimensionality reduction via tangent space alignment," *Journal of Shanghai University (English Edition)*, vol. 8, no. 4, pp. 406–424, 2004.

[209] Zhou, K., Hou, Q., Wang, R., and Guo, B., "Real-time kd-tree construction on graphics hardware," in *ACM Transactions on Graphics (TOG)*, vol. 27, p. 126, ACM, 2008.

INDEX