Poster: Parallel Algorithms for Clustering and Nearest Neighbor Search Problems in High Dimensions

Logan Moon
Institute for Computational Engineering and Sciences
The University of Texas at Austin, Austin, TX
logan@ices.utexas.edu

Daniel Long
School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, GA
dlong3@gatech.edu

Shreyas Joshi
School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, GA
shreyasj@gatech.edu

Vyomkesh Tripathi
School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, GA
vtripathi7@gatech.edu

Bo Xiao
School of Computational Science and Engineering
Georgia Institute of Technology, Atlanta, GA
boxiao33@gmail.com

George Biros
Institute for Computational Engineering and Sciences
The University of Texas at Austin, Austin, TX
gbiros@acm.org

ABSTRACT

Clustering and nearest neighbor searches in high dimensions are fundamental components of computational geometry, computational statistics, and pattern recognition. Despite the widespread need to analyze massive datasets, no MPI-based implementations are available to allow this analysis to be scaled to modern highly parallel platforms. We seek to develop a set of algorithms that will provide unprecedented scalability and performance for these fundamental problems.

Categories and Subject Descriptors:
E.1[Data Structures]:Distributed Data Structures
G.4[Mathematical Software]:Algorithm Design and Analysis
G.4[Mathematical Software]:Parallel and Vector Implementations

General Terms: Algorithms

1. INTRODUCTION

In this poster, we describe message-passing and shared memory parallel algorithms for nearest-neighbor and clustering in Euclidean spaces. Specifically, given a set $\mathcal{R}$ of n *reference points* $\{r_i\}_{i=1}^n \in \mathbb{R}^d$ and a set $\mathcal{Q}$ of m *query points* $\{q_j\}_{j=1}^m \in \mathbb{R}^d$ and defining $d(r_i, q_j) := \|r_i - q_j\|_2$, we consider the following two problems.

- **(1)** *kmeans clustering* (`KMC`):
  Find $\{c_j\}_{i=1}^k$ in $\mathbb{R}^d$ by finding $\{c_j\}$ that minimize $\sum_{j=1}^k \sum_{r_i \in V_j} d(r_i, c_j)^2$, where V_j is the Voronoi set of c_j, $V_j = \{r_i \in \mathcal{R} : c_j = \min_{c_l} d(r_i, c_l)\}$; and
- **(2)** *k- and range nearest neighbor searches* (`KNN`):
  Given a *query point* $q \in \mathcal{Q}$, find the k-nearest neighbors or find all points $r \in \mathcal{R}$ such that $d(r, q) < \rho$, where ρ is the range.

Significance. Nearest neighbor and clustering problems are fundamental problems in computational geometry. They are building blocks for more complex algorithms in computational statistics (e.g., kernel density estimation), spatial statistics (e.g., n-point correlation functions), and machine learning (e.g., classification, manifold learning). In turn, such methods are key components in physics (high-dimensional and generalized N-body problems), dimension reduction for scientific datasets, and uncertainty estimation.

Despite `KNN` and `KMC` methods being fundamental building blocks for many algorithms in computational data analysis, there has not been a lot of work in scaling them to high-performance parallel platforms. There is a rich literature in distributed memory algorithms, particularly in the database community, but the technologies have not yet been migrated to high performance computing platforms.

Related work. For a comprehensive review on clustering methods and computational geometry, see [5] and, for parallel kmeans, see [8]. In [7], the authors present an efficient kmeans algorithm based on a KD-tree data structure; the method was not parallelized. The basic scheme for parallelizing kmeans is simple and is discussed in [3] and [6].

Although there is significant amount of work on indexing structures in the database community (e.g., [2]), and on sequential algorithms [4, 9] for generalized N-body problems and tree data structures, we are not aware of any message passing interface-based scalable algorithms for multidimensional trees. Overall, for both kmeans and k-NN problems, the previous work has been limited to shared memory implementations and distributed memory with quite a small number of processes.

2. METHODS

We have implemented several scalable methods for solving both the k-nearest neighbors and ρ-near neighbors problems. Here, we give an overview of the algorithms and their parallel implementations.

2.1 PCL-Tree Multilevel Partitioning

PCL-Tree is a fully distributed spatial indexing data structure which uses kmeans clustering to recursively partition a data set across MPI processes. Of course, the PCL-

Copyright is held by the author/owner(s).

SC'11 Companion, November 12–18, 2011, Seattle, Washington, USA.
ACM 978-1-4503-1030-7/11/11.

TREE data structure could be used with other clustering or grouping algorithms as well PCL-TREE is suitable for pruning exact and approximate queries on large data sets in which the intrinsic dimensionality is low to moderate. By no means do we claim that our method resolves the "curse of dimensionality" in general.

Tree construction. To enable efficient `KNN` searches in high dimensions, it is necessary to relax the demand for exact searches and combine PCL-TREE with `LSH`. We construct a binary tree in parallel to partition the reference points approximately evenly over p processors using a distributed top-down algorithm to construct the tree and, at each level, split a node to two children, each containing one or more groups of points kmeans clusters. A tree node is shared among multiple processes, each maintaining a local data structure for the tree node containing its MPI communicator and the clustering information for pruning queries. Each process stores these data structures in a doubly-linked list representing a path from root to leaf for that process's portion of the tree. In our implementation, the minimum granularity of a PCL-TREE node is an MPI process.

Limitations. PCL-TREE has several limitations: **(1)** For certain high-dimensional data sets with high intrinsic dimensionality, PCL-TREE is unable to prune points during a query operation; however, this is detectable, allowing for a graceful recovery. **(2)** We have only examined ℓ_2 distance metrics. **(3)** The tree traversals are done exactly. Further approximation is possible using inexact pruning.

2.2 Distributed Direct Search

We have developed two highly scalable parallel algorithms for directly calculating the k-nearest and ρ-near neighbors of bulk sets of query points. In our "rectangular" algorithm, processes are logically arranged in a two-dimensional mesh. The reference and query point sets are then replicated to each row and column of the mesh, respectively. Each process performs a single local search on its partitions of $\mathcal{R}$ and $\mathcal{Q}$. In our "cyclic" algorithm, processes are logically arranged in a ring topology, and $\mathcal{R}$ and $\mathcal{Q}$ are each partitioned approximately evenly across all processes. In each of p iterations, every process performs a local search and then transmits its current portion of $\mathcal{R}$ to the next process.

2.3 Distributed Locality Sensitive Hashing

We have developed a highly scalable distributed `LSH` algorithm based on the hashing scheme proposed in [1]. While we use Andoni and Indyk's hash function families because of their well-established probability of correctness, our overall distributed approach is vastly different:

1. K random hash functions are generated.
2. Each point $r_i \in \mathcal{R}$ is mapped to a bucket using the generated functions.
3. Each bucket is assigned to an MPI process in a manner ensuring a good load balance.
4. Each reference point is transmitted to the process that owns its respective bucket.
5. Each point $q_j \in \mathcal{Q}$ is mapped to a bucket using the same set of functions.
6. Each query point is transmitted to the process that owns its respective bucket.
7. A direct search is performed within each bucket.
8. The results for each query point are collected at a particular process.

The above steps are repeated for a total of L iterations.

3. RESULTS

In our tests, our algorithms show excellent scalability to thousands of MPI processes. In our largest runs, we were able to conduct exact searches and clustering on datasets with over four billion reference points in 1000 dimensions (a 45TB dataset) in under 30 seconds on 98K cores on the NSF NICS Kraken platform. As a second performance highlight, both tree construction and an all-nearest neighbors query take less than two minutes for a dataset with over 240 million points in ten dimensions on 6144 MPI processes. For a 100-dimensional dataset the construction is still fast but due to limited pruning the query takes four to five times longer.

Acknowledgments.

This work was partially supported by the U.S. National Science Foundation grants CNS-0929947, OCI-0749285 and OCI-0749334, the U.S. Air Force Office for Scientific Research grant FA9550-09-1-0679, and the U.S. Department of Energy grant DE-FG02-08ER2585. Computing resources on the XSEDE systems were provided under the grants ASC070050N and MCA04N026. We would like to thank the NCCS and XSEDE support staff, from whom we have received significant assistance. Also, we would like to thank Dr. Andoni of Microsoft Research for giving us the `E`^{2}`LSH` package.

4. REFERENCES

[1] ANDONI, A., AND INDYK, P. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *COMMUNICATIONS OF THE ACM 51*, 1 (2008), 117.

[2] BERCHTOLD, S., KEIM, D., AND KRIEGEL, H. The X-tree: An index structure for high-dimensional data. *Readings in multimedia computing and networking 12* (2002), 451.

[3] DHILLON, I., AND MODHA, D. A data-clustering algorithm on distributed memory multiprocessors. *Large-Scale Parallel Data Mining* (2000), 802–802.

[4] GRAY, A., AND MOORE, A. N-Body'problems in statistical learning. *Advances in neural information processing systems* (2001), 521–527.

[5] JAIN, A., MURTY, M., AND FLYNN, P. Data clustering: a review. *ACM computing surveys (CSUR) 31*, 3 (1999), 264–323.

[6] JOSHI, M. Parallel k-means algorithm on distributed memory multiprocessors. *Computer 9* (2003).

[7] KANUNGO, T., MOUNT, D., NETANYAHU, N., PIATKO, C., SILVERMAN, R., AND WU, A. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2002), 881–892.

[8] OLSON, C. Parallel algorithms for hierarchical clustering. *Parallel computing 21*, 8 (1995), 1313–1325.

[9] RAM, P., LEE, D., MARCH, W., AND GRAY, A. Linear-time algorithms for pairwise statistical problems. *Advances in Neural Information Processing Systems 23* (2009).