A

Parallel Algorithms on Nearest Neighbor Search

BERKAY AYDIN, Georgia State University

The (k-)nearest neighbor searching has very high computational costs. The algorithms presented for nearest neighbor search in high dimensional spaces have have suffered from curse of dimensionality, which affects either runtime or storage requirements of the algorithms terribly. Parallelization of nearest neighbor search is a suitable solution for decreasing the workload caused by nearest neighbor searching in high dimensions.

In this paper, we have surveyed notable advancements in the parallelization efforts on nearest neighbor searching algorithms. The algorithms covered are ordered chronologically for exhibiting the evolution throughout the years. Recent improvements on GPU based nearest neighbor search algorithms are covered. Additionally, we address the open problems in the area of parallel nearest neighbor algorithms.

Contents

1 Introduction **2**
- 1.1 Contributions . . . 3
- 1.2 Scope . . . 3
- 1.3 Organization of the Paper . . . 3

2 Preliminary Concepts and Definitions **3**
- 2.1 Near Neighbor Queries . . . 3
- 2.2 Nearest Neighbor Queries . . . 4
- 2.3 k-Nearest Neighbor Queries . . . 4
- 2.4 Finding Nearest Neighbors . . . 4
  - 2.4.1 Increasing Range . . . 4
  - 2.4.2 Decreasing the Radius and Backtracking . . . 4

3 Foundation of Parallelization in Nearest Neighbor Search **4**

4 Evolution of Parallel Nearest Neighbor Search **6**

5 Recent Nearest Neighbor Search Algorithms Using GPUs **9**
- 5.1 Brute Force GPU Implementations . . . 9
- 5.2 Kd-Tree Based GPU Implementations . . . 10
- 5.3 Locality Sensitive Hashing Based GPU Implementations . . . 11

6 Open Problems and Conclusion **12**

This work is done for CSC 8530 Parallel Algorithms class at the Department of Computer Science in Georgia State University. This project was supervised by Dr. Sushil Prasad.
Author's address: B. Aydin, Department of Computer Science, Georgia State University, 34 Peachtree St Suite 1450 Atlanta, GA 30303.

DOI:http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

Searching is one of the fundemental problems in computer science field. Most of the early searching algorithms were proposed for the equality-based search, where the data points are well structured. More advanced algorithms for searching were inequality based (or even more sophisticated, range queries), where the data points can be ordered based on an ordering operator. Traditional databases have supported such queries for long time; however, with the advancements in the information retrieval systems, the focus of searching has been changed to similarity search. The interest to similarity search arised from the high dimensional data which is extracted from different data sources that emerged in the end of 1990s (e.g. images, sounds, time sequences). Similarity search, also called proximity search, is searching for data points that are close to a given query point based on a distance measure.

For high dimensional data, because dimensions are very high (as name suggests), volume of the space for data points increases tremendously, as they become more sparse with more dimensions [Berchtold et al. 1997]. Moreover, there is no natural ordering among the dimensions, where we could hierarchically search using classical equality based techniques. Under these conditions, similarity based search became more popular, since equality-based or range queries do not suffice the needs of different fields such as data mining, computational geometry, bioinformatics etc.

Nearest neighbor search is one type of similarity search which can be formalized as follows. Given d-dimensional space $\mathbb{R}^d$, a query point $q \in \mathbb{R}^d$, and a set of data points P, the *nearest neighbor* query returns the data point in P that is closest to the q with respect to a defined distance measure $\mathcal{D}$. There are a variety of distance measures for different data topologies and characteristics. The *closeness* property is dependent on the distance measure.

One particular type of nearest neighbor search algorithms is finding k-nearest neighbors (shortly kNN). k-nearest neighbor queries have been utilized by a large spectrum of algorithms. First of all, information retrieval systems, with the focus on high dimensional data is the main application area for nearest neighbor search. A wide range of search engines either for text, image or audio retrieval uses nearest neighbor queries for retrieving the data as well as ranking them. Aside from pure searching applications, one common application domain for nearest neighbor search is instance based learning algorithm called kNN classifier. kNN-based clustering techniques are also proposed. Since finding kNN of a query point is a computationally expensive task, approximation techniques for k-nearest neighbor are also presented [Liu et al. 2004]. The approximate nearest neighbor queries return points that are at most c times the distance from the query to its nearest points. (c is also referred as approximation factor.)

Other than the approximation techniques, parallelization of nearest neighbor searching algorithms have also been presented throughout the last 25 years. The challenge of nearest neighbor algorithm is the high dimensions affecting the runtime or space complexity of the proposed search algorithms. This particular characteristic of nearest neighbor search makes it very suitable for using parallel algorithms. Most of the sequential algorithms require accessing and manipulating the distinct data regions, either for forming the data structure or retrieving the data. Throughout the last 25 years, researchers working on nearest neighbor queries have shown that popular data structures such as R-trees and its variants, as well as new class of hashing and clustering-based techniques can be parallelized. In addition to that, in recent years general purpose GPU programming has been utilized for nearest neighbor searching.

1.1. Contributions

In this paper, a broad range of the parallel nearest neighbor and k-nearest neighbor algorithms have been inspected. To our knowledge there are no survey papers exhibiting a comprehensive investigation on parallel nearest neighbor algorithms. Our contributions are as follows: (1) The early papers on the foundation of parallel nearest neighbor search algorithms have been shown. (2) The evolution of these algorithms have been inspected (3) GPU implementations for nearest neighbor search algorithms (which are massively parallel) have also been investigated and and they are classified based on their methods of parallelization. (4) Open problems in this research area have been presented.

1.2. Scope

The scope of this paper is parallel nearest neighbor algorithms with the focus on k-nearest neighbor search algorithms. Recent GPU-based implementations are demonstrated in the paper, as they are related to parallelization efforts of nearest neighbor queries. One type of the similarity search is the range queries. However, this paper does not cover most of the range query searching algorithms; we only cover the ones that are particularly proposed for finding k-nearest neighbor queries. In addition to GPU-based implementations, before concentrating on parallel nearest neighbor search, some preliminary concepts and definitions on nearest neighbor search are shown to introduce the topic to reader; nevertheless, the classification of the types of similarity search and the sequential algorithms to retrieve the nearest neighbor queries are not in the scope of this paper. Finally, our scope is limited to metric spaces, and mostly vector spaces, similarity search in non-metric spaces are not in the scope of this paper.

1.3. Organization of the Paper

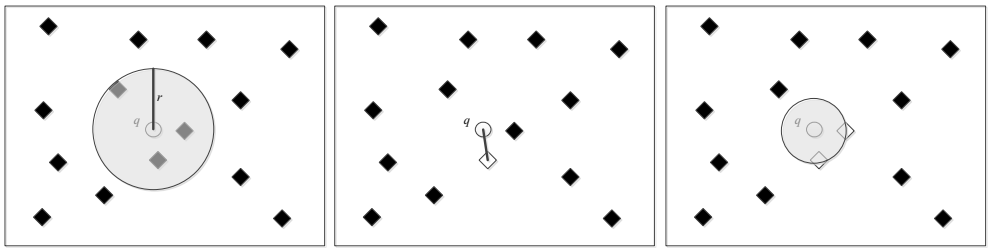
Rest of this paper is organized as follows. In Section 2, we introduce the preliminary concepts on similarity search and nearest neighbor search. In Section 3, early parallelization efforts on the nearest neighbor searching have been investigated. Foundational papers on the problem are presented. Then, the evolution of the nearest neighbor algorithms have been shown in Section 4. Later, GPU based algorithms on parallel nearest neighbor algorithms have been investigated in Section 5. Lastly, open problems presented and summary is given in Section 6.

2. PRELIMINARY CONCEPTS AND DEFINITIONS

In this section, the preliminary concepts on similarity search will be demonstrated. In metric spaces, for most cases, there are three types of similarity search that can be performed. The first one is the range queries, which can also be classified as near neighbor queries. Second one is the nearest neighbor query, where the closest data point to the query is to be retrieved. Third case is the k-nearest neighbor query that is a specialized version of nearest neighbor queries where we want to get k-closest elements to the query point. All three of these query types can be used to find another using approximation techniques. Additionally, we cover methods for finding nearest neighbors.

2.1. Near Neighbor Queries

Near neighbor query (range query) is described as follows: Given a query point q and a distance r, we want to return the elements that are within the distance of r to the query point q. For most cases, near neighbor queries are less expensive to run and retrieve. Because of that, they have been used as a filtering mechanism for finding more com-



(a) Near Neighbor Query (b) Nearest Neighbor Query (c) 2-Nearest Neighbor Query

Fig. 1. Types of similarity search in 2-dimensional metric space using Euclidean metric.

plicated queries. An example illustration for near neighbor queries in 2-dimensional space using Euclidian distance metric is shown in Figure 1.a.

2.2. Nearest Neighbor Queries

Nearest neighbor search is one of the earliest proximity (similarity) search problems. Simply, this type of queries return the closest element to the query point. An example of such a query can be seen in Figure 1.b.

2.3. *k*-Nearest Neighbor Queries

In k-nearest neighbor queries, we fetch k data points that are closest to the given query point. In Figure 1.c, we show an example of k-nearest neighbor query where $k = 2$. Nearest neighbor queries are the specialized version of k-nearest neighbor queries, where k is set to 1. As discussed earlier, range (near neighbor) query is the simplest query type, and it is altered and manipulated for finding the other two types of queries.

2.4. Finding Nearest Neighbors

Without considering the exceptions in the literature, there are two fundamental techniques to explore the k-nearest neighbors of a query point [Chavez et al. 2001]. It is necessary to note that nearest neighbor queries are built on top of range queries.

2.4.1. Increasing Range. The most intuitive way for finding nearest neighbors is done by querying a region with a predetermined range. For a given range, we firstly check if k elements are found within that range; if not we send a range query with an increased range parameter. This process is repeated until at least k elements fall into the desired hypersphere.

2.4.2. Decreasing the Radius and Backtracking. In this type of nearest neighbor searching algorithms, a data structure or a clustering methodology is needed for efficient range decreasing. We start querying (range query) with a large range (or maximum distance between two elements) and keep the k-nearest elements found; then, pick another candidate point and apply range search with decreased radius. Each time the range decreased, we update k-nearest points.

The backtracking idea can be prioritized with tree-based data structures that partitions the space efficiently. The search is followed by heuristics for picking smarter canidate points, that are closer to query point with higher probabilities.

3. FOUNDATION OF PARALLELIZATION IN NEAREST NEIGHBOR SEARCH

The dimensions in of early similarity search apllications are not extremely high as it is now. First attempts to parallelize the nearest neighbor search was mostly field based. Those fields were computaional geometry, bioinformatics, spatial databases and access methods.

The first known parallel algorithm to our knowledge created for answering the nearest neighbor queries was done by Goodrich [Goodrich 1985]. The presented algorithm finds the nearest neighbors of all vertices in a convex polygon (all-nearest-neighbor). This algorithm runs in $O(logn)$ time with $O(n/logn)$ processors. Concurrent-Read Exclusive-Write (CREW) Prallel Random Access Machine (PRAM). Thu knowledge of polygon being convex have been used in their algorithms and the algorithm consists of a partition step and a merge step. Even though this algorithm heavily depends on the convexity of polygons, especially in partition phase, it is the first known parallel algorithm in the field.

In [Dehne et al. 1993], a set of useful algorithms for computational geometry field were proposed. Another parallel algorithm for finding the nearest neighbor in 2-dimensional space is presented, but this time without the restriction (or knowledge) of convex polygons. Coarse grained multicomputer model (where there are p processors and each processor has $O(n/p)$ local memory) have been used. Their algorithm for nearest neighbor finding is based on distributing the interesting points (nearby) partially to different processors. First, the algorithm sorts the points by x-coordinates, then 1-dimensional all-nearest-neighbor finding is performed. Later, same procedure applied for y-dimensions. Then, the results are broadcasted and 2-dimensional nearest neighbors are found and merged. Their algorithm runs in $O(nlogn/p + T_s)$ time, where T_s shows the sorting time.

Dynamic programming have also been used for similarity search, and the parallelization is applied to dynamic programming method for similarity search and sequence alignment [Galper and Brutlag 1990]. The dynamic programming method is used for edit distance in biological sequences and a shared-memory architecture is utilized. Several decompositions of value calculations in dynamic programming method were discussed, and they were able to get a significant speed-up using multiple processors. This method is simply based on the range search on sequence data; however, it is included in this survey, because it is one of the first attempts in similarity search parallelization.

Callahan presented an optimal algorithm for all-nearest-neighbors queries using well-separated pair decomposition in [Callahan 1993]. How this algorithm can be tuned for k-nearest neighbor queries was also demonstrated. The model used was concurrent-read exclusive write (CREW) PRAM. The notion for this particular algorithm is the well-separated pair decomposition and it is done by using a fair split tree. Parallelization is applied for constructing the fair split trees and retrieving the pairs (which will later on be used for getting the nearest neighbors). Fair split tree is an extension to quadtrees and it is a space-partitioning tree. The fair split tree for d-dimensional case can be constructed in $O(logn)$ time and $O(n)$ processors. Computing well-separated pairs takes $O(logn)$ time with $O(n)$ processors. Therefore, $O(logn)$ time is achieved for retrieving nearest neighbors. (Note that the author did think of k as a constant.)

It is also important to note the indexing structures that has been used for finding nearest neighbor queries. R-trees are introduced by Guttman in 1984 are still one of the most popular indexing structures for spatial and spatiotemporal access methods, and have inspired many different indexing structures [Guttman 1984]. Kamel and Faloutsos introduced the parallelized version of R-trees [Kamel and Faloutsos 1992]. In their architecture data is distributed over different disks. They propose two distributions of data, one is in round robin fashion, while the other one is done using partitioning the space. Then, an R-tree structure built for each disk. Moreover, they propose another approach, by using a different node structure called *supernode*. Each supernode references pages that are distributed over different disks. Later, another parallel R-tree structure, *MX-R-tree* is presented by same authors in [Kamel and Faloutsos

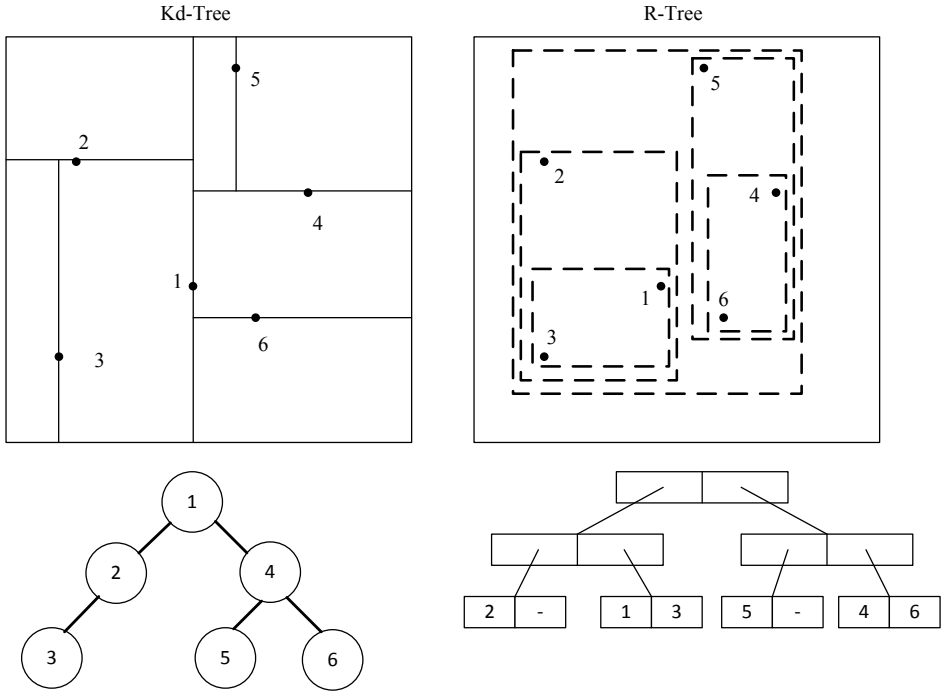


Fig. 2. Space-based partitioning in Kd-trees and data-driven partitioning of R-trees

1993]. In MX-R-tree, the root node is kept in main memory. Inner nodes representing monimum bounding rectangles span one disk page and they are distributed over different disks. These data structures can answer nearest neighbor queries very well; however, as discussed earlier with higher dimensions trees become more sparse and memory space grows exponentially. Same problem applies for kd-tree, which is another space partitioning method, that has been widely used for answering nearest neighbor queries. Kd-trees are k-dimensional binary search trees and they were introduced in [Bentley 1975]. Several parallel implementations of kd-trees were presented recently. They will be discussed in Section 5. The space partitioning methodology for R-trees and kd-trees can be seen in Figure 2

4. EVOLUTION OF PARALLEL NEAREST NEIGHBOR SEARCH

With the advancements in the technology in late 1990s and early 2000s, the data collected from various sources started to become massive. Moreover, massive increase in the dimensions of collected data could also be observed. Image, audio and video retrieval and recognition, processing and retrieving time series data, huge text databases were (and still are) some representatives of the common applications that transformed the nearest neighbor searching to a more challenging problem. In all these aforementioned applications, converting the data to interesting and more meaningful feature vectors are necessary.

As a consequence of the extraction of features, elevated number of dimensions appeared more frequently, and *curse of dimensionality* became a remarkable problem to overcome. (In simple terms, the exponential increase with recpect to number of dimensions in the memory requirements and/or runtime requirements caused by high dimensions is referred as curse of dimensionality.) Initial algorithms for similarity

search problem mostly did not address a practical solution for actually high dimensional data. However, later papers, both sequential and parallel, are actually aiming to overcome the curse of dimensionality. In this section, parallel approaches for overcoming curse of dimensionality problem will be investigated.

In [Papadopoulos and Manolopoulos 1997], a shared-nothing parallel architecture is presented for nearest neighbor queries. A static data distribution policy using fractal curves (in this paper, it is Hilbert curve) is employed. The primary server indexes the complete static directory, while secondary servers indexes the data pages. For k-nearest neighbors query, a depth-first search in the primary server is used (for accessing data pages stored in secondary servers), then a sphere around the lowest level of primary index is decided and multiple regions are activated and searched simultaneously.

Berchtold et al. proposed a parallel similarity search algorithm that is designed for high-dimensional spaces [Berchtold et al. 1998]. They identify the essence of the nearest neighbor search in high dimensional as distributing data over different disks *evenly*, therefore, it can be processed or retrieved efficiently. Their algorithm is based on an efficient clustering method, which distributes the data over disks using a bit string that encodes the quadrant that data point lies. The bit string is simply processed using bitwise XOR operation. The reason for using XOR operation is to handle partial matches. Using that schema, the memory space required for nearest neighbor search becomes linear in terms of number of dimensions. An extension to their distribution schema, where arbitrary number of disks are sufficient for the algorithm is also discussed.

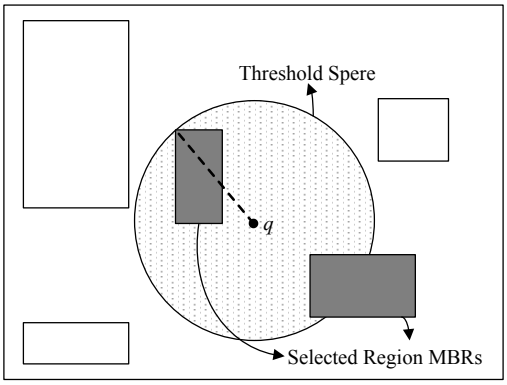


Fig. 3. An illustration of threshold sphere used in [Papadopoulos and Manolopoulos 1998] as a filtering mechanism

In [Papadopoulos and Manolopoulos 1998], two new algorithms are proposed for nearest neighbor query processing using multiple disks. In R-trees sibling nodes contained in a parent node are generally closer to each other than the non-sibling nodes. Their method utilizes a proximity index, where the selected disk for a node (storing the minimum bounding rectangle of some points) must contain least proximal siblings of the node. Therefore, the disk pages contain furthest siblings. One of two algorithms presented is called full parallel similarity search (FPSS). In FPSS, a threshold sphere is used as a filtering mechanism. Threshold sphere is bounded by the maximum distance between query point and the closest node's minimum bounding rectangle (An illustration of 2-dimensional threshold sphere can be seen in Figure 3). After filtering, all the remaining disks (that are not pruned by threshold sphere) are searched in

parallel, and the results are refined for finding the nearest neighbors. The more sophisticated second algorithm is called candidate reduction similarity search (CRSS), where the search regions after filtering step (same with FPSS, threshold sphere) are processed in a smarter way. The minimum bounding rectangles that are within the threshold sphere are given more priority than the ones that are only intersecting but not contained.

In [Braunmuller et al. 2001], a schema for processing multiple k-nearest neighbor queries is proposed. The motivation for this paper is derived from data mining techniques that require both I/O and CPU intensive operations. The approach is mainly utilized for reducing the I/O or CPU cost by decreasing the number of disk accesses via pre-computation of the distance between multiple query points. Later, the closer query points are grouped and a batch query is issued, which (according to their experiments) lead to a significant decrease in total I/O cost. Pivot-based pruning mechanism is used as a filtering mechanism in this paper. The distances of data points to pivots are precomputed and techniques for sending multiple queries are investigated as a way of parallelization of similarity queries.

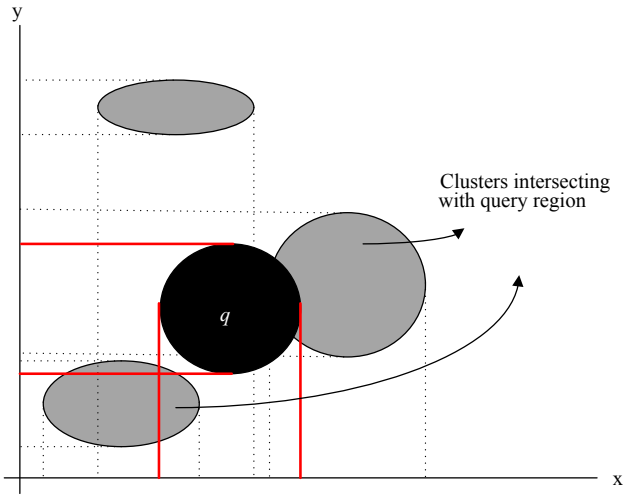


Fig. 4. Searching a region in PN-Tree [Ali et al. 2005]

Projected Node Tree (PN-Tree) presented in [Ali et al. 2005] is a parallel multidimensional indexing technique that can be used for processing nearest neighbor queries efficiently. PN-tree depends on the clustering of data points. Mapping high dimensions to B+ trees is a novel idea that is used in this paper. PN-tree firstly clusters the data points into high dimensional nodes, then clustered points are projected over each dimension and all projections indeced using a separate B+ tree. The index structure has a set of one-dimensional index structures and a small multidimensional indexing structure for keeping the centroids of clusters. Search is performed in parallel over B+ trees. Search region is also projected over each dimension and all the B+ trees searched in parallel. Figure 4 demonstrates a query region intersecting with two clusters in 2-dimensional space.

5. RECENT NEAREST NEIGHBOR SEARCH ALGORITHMS USING GPUS

In the previous sections, the parallel algorithms for nearest neighbor searching are covered. Recent trend in the research for parallelization of nearest neighbor search is to use graphics processing units (GPU) to answer a similarity search query. General purpose GPU programming became popular with the creation of heteregenous parallel programming and computing platforms such as NVIDIA's Compute Unified Device Architecture (CUDA) [1] and Open Computing Language (OpenCL) [2]. The nearest neighbor searching problem also slithers through the usage of GPUs. The programming model that has been employed in GPU programming is suitable for certain tasks in nearest neighbor search; however, the data transfer time between central processing unit (CPU) and GPUs is still a bottleneck as nearest neighbor search algorithms can be both I/O and CPU intensive. (CPU intensiveness is caused by the high computation load while finding the distances, some distance measures such as non-metric KLD (Kullback-Leibler divergence) and Hausdorff distance are computationally very expensive for large datasets.)

The early efforts on employing GPUs for nearest neighbor searching is presented in [Bustos et al. 2006], [Garcia et al. 2008], [Liang et al. 2009], [Kuang and Zhao 2009], [Liang et al. 2010] and [Garcia et al. 2010]. The earlier algorithms used brute force method for finding the nearest neighbors, and they observed significant increase in the runtime because of the massive computational power via parallelization enabled by GPUs. These brute force algorithms further investigated in [Kato and Hosino 2010] and [Sismanis et al. 2012]. Later , we observe the new advancements for the utilization of GPUs. Such improvements are introduced in more recent papers [Pan et al. 2010], [Pan and Manocha 2011], [Cayton 2012]. Those techniques are based on different data structures such as locality sensitive hashing, random ball cover and hash index on GPUs. In addition to these, kd-tree based approaches are also pretty popular; and some of them are presented in [Zhou et al. 2008], [Qiu et al. 2009], [Hering 2013], [Gieseke et al. 2014].

5.1. Brute Force GPU Implementations

Sequential scan is one of the most commonly used techniques for performing similarity search. Many of the nearest neighbor search algorithms. A comprehensive analysis shown that for (very) high dimensionality the nearest neighbor search becomes inherently linear for a variety of data distributions [Beyer et al. 1999]. Therefore, performing sequential (linear) scan for finding nearest neighbors is an appropriate heuristic, even though it lacks novelty.

The GPU algorithms that perform sequential scan (linear search) shares common properties. In most cases, there are two operations: (1) finding the distances between data points and (2) sorting them in increasing order or getting the maximum. The brute force algorithms differ in two aspects: distance measure and sorting algorithm.

The first appearance of similarity search implementation using GPUs is demonstrated in [Bustos et al. 2006]. This algorithm finds the nearest neighbor ($k = 1$ for kNN). The distance measure employed is $Manhattan$ distance. The results show that GPU-based implementation provides 4 to 6 times increase in runtime and linear scalability for database size and number of dimensions are also provided.

Garcia et al. presents a brute force k-nearest neighbor search algorithm using GPUs [Garcia et al. 2008]. Distance measure used is KLD. They use comb sort (a variation of quicksort) and insertion sort. The rationale given for using insertion sort is that only k neighbors need to be retrieved for kNN queries. In their settings (4800 data

[1] http://www.nvidia.com/cuda
[2] https://www.khronos.org/opencl/

points with 64 dimensions), insertion sort beats the comb sort for k values less than 110. It is important to note that, runtime for insertion sort increases linearly while comb sort one stays constant. In the follow up work [Garcia et al. 2010], they compare their extended technique using CUBLAS library (CUDA implementation from BLAS[3]) to popular approximated nearest neighbor library ANN[4] for C++.

In [Liang et al. 2009], a similar approach is used with Garcia et al.'s method for finding k-nearest neighbors. The sorting techniqe employed is quicksort. They implement two kernels for distance computation and sorting. They get more than 45 times speedup compared to a serial kNN algorithm. In their follow-up paper[Liang et al. 2010], they introduce another dimension to problem by presenting a way to answer multiple kNN queries. Simply, for multiple queries a new kernel for distance calculation and sorting is started for each thread and for r queries r distances are calculated and stored; then each group of distances is sorted in parallel.

In [Kuang and Zhao 2009], CUDA-based kNN algorithm is proposed, just as its predecessors. This time radix sort is employed, the distance is calculated using Euclidean distance measure.

In [Kato and Hosino 2010], a kNN searching solution for the multiple GPUs is given. They observe the kNN problem as N-body problem. Their algorithm includes partial sort of multiple arrays of distances. To decrease the synchronization cost, they introduce a partial sorting technique for kNN problem. For that, they keep a heap structure of size-k (at most). Each GPU keeps n heaps for n data points it stores, and all heaps store k-nearest neighbor elements computed with itself. Later, the heaps from different GPUs merged in the last phase.

In a more recent work [Sismanis et al. 2012], a somewhat more novel brute force algorithm has been proposed. They use truncated bitonic sort as their sorting technique and they claim and show that it works better than the earlier select-and-sort algorithms. They achieve better performances using their truncated bitonic sort compared to radix sort based implementations.

5.2. Kd-Tree Based GPU Implementations

Kd-trees have been used for nearest neighbor searching for long time. In kd-trees, data point insertions split the space, and each split is done on a different dimension. For similarity search in kd-trees, a procedure similar to regular search is performed and the closer leaf points are fetched sequentially. The original kd-trees, being one of the most popular data structures for similarity search, are affected heavily from the curse of dimensionality. For that reason, variations on kd-trees are implemented on GPUs.

In [Zhou et al. 2008], an algorithm that shows the real time kd-tree construction is presented for GPUs. Their version of kd-trees build the nodes in breadth-first search fashion. Moreover, larger nodes in the higher levels of the tree are used. Fast evaluations for split costs are also presented. The kNN implementation is based on a range search on the tree (with a given radius), and it continues to increase the size of the radius until k elements are retrieved.

In [Qiu et al. 2009], a nearest neighbor search algorithm based on kd-trees proposed for 3D registration. A priority queue based approach is used for searching for the nearest neighbors. Their results show that for 3D registration they achive 86 times faster performance using GPUs.

In the very recent work [Gieseke et al. 2014], a buffer kd-tree for GPUs is presented. The *buffer* refers to a query buffer located in every node of kd-tree, which is used to

[3]https://developer.nvidia.com/cublas
[4]http://www.cs.umd.edu/ mount/ANN/

delay the execution of queries. Each node in the buffer kd-tree corresponds to a set of reference patterns. Therefore, a lazy nearest neighbor search schema is applied. The authors state that their querying schema is suitable for batch queries or for the ones that does not require immediate response. The dimensions considered in the experiments are less than 25. The speedups between 15 to 22 times are obtained using buffer kd-tree.

5.3. Locality Sensitive Hashing Based GPU Implementations

Locality sensitive hashing is introduced in [Indyk and Motwani 1998], as a solution to approximate nearest neighbor problem. It is a hashing based indexing structure that clusters the data points to the closer hash buckets by using multiple probabilistic hash functions and storing them in different hash tables (An illustration is shown in Figure 5). Theoretically, locality sensitive hashing based techniques remove the curse of dimensionality, the exponential growth in the runtime or memory by number of dimensions are removed. However, choosing the right approximation factor becomes very important, as the total complexity of the problem depends on the approximation factor.

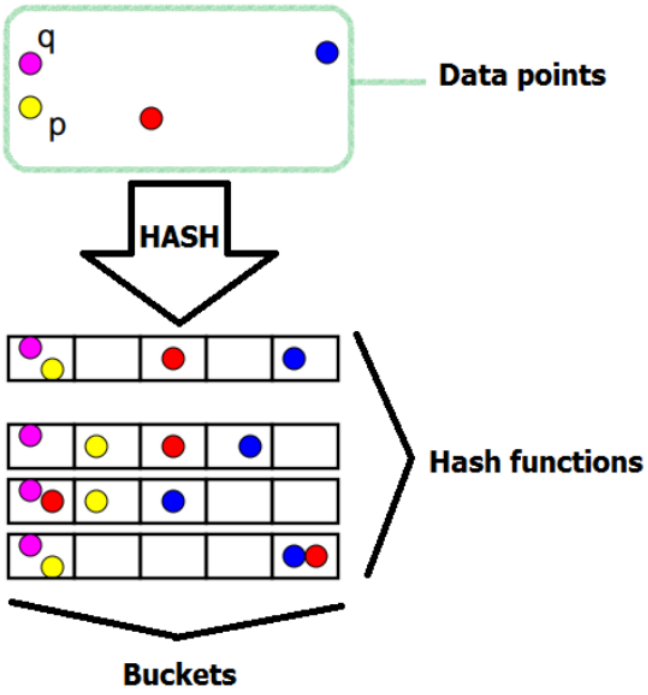


Fig. 5. Locality sensitive hashing schema described in [Indyk and Motwani 1998]

Some recent papers introduced LSH based similarity searching algorithms using GPUs. For LSH requires the computation of multiple probabilistic hash functions. These functions simple will place the closer data points to the closer (hopefully same) buckets with much higher probability. To improve the quality of hashing schema, more hashing functions are supposed to be used. This heavy calculation step requires high computational resources and it can be parallelized using GPUs.

In [Pan et al. 2010], LSH based k-nearest neighbor search on GPUs are proposed, and it is particularly implemented for robot motion planning algorithms. The LSH computations are altered for *weighted* Euclidean space. Parallelization method used is established upon sending batch queries to the indexing structure, so essentially each thread computes LSH value for one query point. Then, each thread finds the k-nearest neighbor in the retrieved buckets. In the successor paper of the same authors, the LSH schema is extended with bi-level LSH computation [Pan and Manocha 2011]. The first level partition the data points using RP-tree (random projection tree, is a data structure relies on space-driven partitioning). The second level is the computation of LSH values. They obtained significant increases on runtime (40 times better) on their experiments, in which they have used 200,000 images represented as 512 dimensional vectors.

6. OPEN PROBLEMS AND CONCLUSION

We have covered many different algorithms proposed for parallel similarity search. The outline of this survey follows the chronological ordering and we wanted to show classical solutions, the improvements on classical solutions and the current trends in the field of parallel similarity search. We have covered notable research papers exhibiting novel solutions for nearest neighbor search. As range search (near neighbor queries such as locality sensitive hashing), nearest neighbor search and k-nearest neighbor search can be used for reduction of each other, our survey contains representative parallelization techniques from all of those three types of similarity search. However, we have tried to keep our focus on k-nearest neighbor searching as it is the most computationally costly one among those three.

Our survey started with early parallelization efforts on parallelizing nearest neighbor search in Section 3. We observe that the motivation for the algorithms presented until 1995 are coming from particular research domains that are not limited to information retrieval and database fields. These research fields include robot motion planning, computational geometry and spatial access methods etc. It is also important to note that most of these algorithms are actually not high dimensional. Therefore, they do not even mention the curse of dimensionality as a problem.

Secondly, we have covered the evolution of early algorithms in Section 4. With the advancements on emerging research fields such as machine learning, pattern recognition and data mining, we start seeing parallel algorithms that are essentially addressing the high dimensionality in nearest neighbor searching. Those fields require the processing of massive as well as high dimensional data. The data collection from different sources and multimedia feature extraction for image, audio and video recognition can be seen as the reasons for the increasing dimensionality. Clustering-based and indexing-based techniques have been proposed. Another pattern we see is the idea of distrubuting the work load to different machines. This lead to clustering to become more effective solution. Moreover, systems that provide results for multiple queries (or simply batch querying) are also presented. This can be explained with the growth of databases that can be accessed using Internet (for instance for image search).

Lastly, we have exhibited recent GPU-based parallel algorithms for nearest neighbor searching in Section 5. In recent years, because usage of graphics processing units has become very popular for solving computationally heavy tasks, the information retrieval community have also used them for similarity searching. There, we investigated the papers that shows how to utilize GPUs for brute-force similarity search. Even though, the brute force solutions seem not novel, using brute-force algorithm is an appropriate idea. One reason for that is sequential search does not require massive memory or storage requirements. Considering the limited memory in GPUs and high cost of transmitting the data between CPU and GPU, using brute force similarity search algorithms is

a very suitable method. The brute-force algorithms presented mostly differ in the distance measures that are calculated among data points and the sorting method utilized for finding k-nearest neighbors. Kd-tree based approaches are also presented. Buffer kd-tree is used for answering batch queries by grouping the queries based on localization. Finally, we have demonstrated the locality sensitive hashing based techniques for finding k-nearest neighbors.

It is also very important to note open problems in the parallel similarity search. The motivation behing the algorithms presented for k-nearest neighbor searching or range searching is mostly based on the problems arising in several other domains. However, a robust similarity searching algorithm that can work universally for multiple purposes of different domains is not yet proposed. High dimensional indexing techniques is a large branch and some classical techniques such as Pyramid, iMinMax and iDistance are suggested for tackling this problem. Locality sensitive based hashing is another high dimensional indexing technique; however, it does not perform well in real life applications. Parallelizations of high dimensional indexing techniques is still an open problem for nearest neighbor searching; especially, for iDistance which can be tuned for answering nearest neighbor queries. In addition to that, clustering-based techniques are also not parallelized fully. The main reason for those is the storage requirements of these high dimensional indexing and clustering techniques. Therefore, reducing the dimensionality is one future direction for nearest neighbor search. Nevertheless, it may not be feasible to use dimensionality reduction techniques, then distributed or cloud based systems for answering nearest neighbor queries can be another research direction to canalize.

There is definetely room for improvement in the parallel similarity search algorithms. Near-optimal algorithms have been proposed for similarity search in sequential settings with locality sensitive hashing. However, most of the recent work done on graphics processing units do ot consider the optimality. The recent parallelization efforts using GPUs are mostly pragmatical; they search the nearest neighbors by utilizing all possible resources mostly in a sequential way. However, the experimental settings that are used in the papers do not push to the limits of data transfers on GPUs. Finding an optimal algorithm as well as removing the curse of dimensionality in parallel settings have not yet been an enlightened area and both stay as an open problem.

Another major shortage to note for similarity search algorithms is the lack of a proper benchmark for high dimensional nearest neighbor search. Throughout the years, almost every research group used their own datasets of interest to test their algorithms. Additionally, the hardware that is being used changed throughout the years. Because of that, the efficiency of the algorithms presented cannot be compared easily.

REFERENCES

Mohamed H Ali, Amani A Saad, and Mohamed A Ismail. 2005. The PN-tree: A parallel and distributed multidimensional index. *Distributed and Parallel Databases* 17, 2 (2005), 111–133.

Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517. DOI:http://dx.doi.org/10.1145/361002.361007

Stefan Berchtold, Christian Böhm, Daniel A Keim, and Hans-Peter Kriegel. 1997. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 78–86.

Stefan Berchtold, Bernhard Ertl, Daniel A Keim, H-P Kriegel, and Thomas Seidl. 1998. Fast nearest neighbor search in high-dimensional space. In *Data Engineering, 1998. Proceedings., 14th International Conference on*. IEEE, 209–218.

Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is nearest neighbor meaningful? In *Database TheoryICDT99*. Springer, 217–235.

Bernhard Braunmuller, Martin Ester, H-P Kriegel, and Jorg Sander. 2001. Multiple similarity queries: A basic DBMS operation for mining in metric databases. *Knowledge and Data Engineering, IEEE Transactions on* 13, 1 (2001), 79–95.

Benjamin Bustos, Oliver Deussen, Stefan Hiller, and Daniel Keim. 2006. A Graphics Hardware Accelerated Algorithm for Nearest Neighbor Search. In *Computational Science ICCS 2006*, VassilN. Alexandrov, GeertDick Albada, PeterM.A. Sloot, and Jack Dongarra (Eds.). Lecture Notes in Computer Science, Vol. 3994. Springer Berlin Heidelberg, 196–199. DOI:http://dx.doi.org/10.1007/11758549_30

Paul B Callahan. 1993. Optimal parallel all-nearest-neighbors using the well-separated pair decomposition. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*. IEEE, 332–340.

Lawrence Cayton. 2012. Accelerating nearest neighbor search on manycore systems. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*. IEEE, 402–413.

Edgar Chavez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in Metric Spaces. *ACM Comput. Surv.* 33, 3 (Sept. 2001), 273–321. DOI:http://dx.doi.org/10.1145/502807.502808

Frank Dehne, Andreas Fabri, and Andrew Rau-Chaplin. 1993. Scalable parallel geometric algorithms for coarse grained multicomputers. In *Proceedings of the ninth annual symposium on Computational geometry*. ACM, 298–307.

Adam R Galper and Douglas L Brutlag. 1990. *Parallel similarity search and alignment with the dynamic programming method*. Knowledge Systems Laboratory, Medical Computer Science, Stanford University.

Vincent Garcia, Eric Debreuve, and Michel Barlaud. 2008. Fast k nearest neighbor search using GPU. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*. IEEE, 1–6.

Vincent Garcia, Eric Debreuve, Frank Nielsen, and Michel Barlaud. 2010. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE, 3757–3760.

Fabian Gieseke, Justin Heinermann, Cosmin Oancea, and Christian Igel. 2014. Buffer kd Trees: Processing Massive Nearest Neighbor Queries on GPUs. In *Proceedings of The 31st International Conference on Machine Learning*. 172–180.

Michael T Goodrich. 1985. An Optimal Parallel Algorithm For the All Nearest–Neighbor Problem for a Convex Polygon. *Computer Science Technical Reports* (1985), 453–463. http://docs.lib.purdue.edu/cstech/453

Antonin Guttman. 1984. R-trees: A Dynamic Index Structure for Spatial Searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD '84)*. ACM, New York, NY, USA, 47–57. DOI:http://dx.doi.org/10.1145/602259.602266

Tim Hering. 2013. Parallel Execution of kNN-Queries on in-memory KD Trees.. In *BTW Workshops*. 257–266.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.

Ibrahim Kamel and Christos Faloutsos. 1992. Parallel R-trees. *SIGMOD Rec.* 21, 2 (June 1992), 195–204. DOI:http://dx.doi.org/10.1145/141484.130315

Ibrahim Kamel and Christos Faloutsos. 1993. High Performance R-trees. *IEEE Data Eng. Bull.* 16, 3 (1993), 28–33. http://sites.computer.org/debull/93SEP-CD.pdf

Kimikazu Kato and Tikara Hosino. 2010. Solving k-nearest neighbor problem on multiple graphics processors. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 769–773.

Quansheng Kuang and Lei Zhao. 2009. A practical GPU based KNN algorithm. In *In Proceedings of the Second Symposium on International Computer Science and Computational Technology (ISCSCT 09) (Dec. 2009), Academy Publisher*. 151–155.

Shenshen Liang, Ying Liu, Cheng Wang, and Liheng Jian. 2009. A CUDA-based parallel implementation of K-nearest neighbor algorithm. In *Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009. CyberC'09. International Conference on*. IEEE, 291–296.

Shenshen Liang, Ying Liu, Cheng Wang, and Liheng Jian. 2010. Design and evaluation of a parallel k-nearest neighbor algorithm on CUDA-enabled GPU. In *Web Society (SWS), 2010 IEEE 2nd Symposium on*. IEEE, 53–60.

Ting Liu, Andrew W Moore, Alexander G Gray, and Ke Yang. 2004. An Investigation of Practical Approximate Nearest Neighbor Algorithms.. In *NIPS*, Vol. 5. 32–33.

Jia Pan, Christian Lauterbach, and Dinesh Manocha. 2010. Efficient nearest-neighbor computation for GPU-based motion planning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2243–2248.

Jia Pan and Dinesh Manocha. 2011. Fast GPU-based locality sensitive hashing for k-nearest neighbor computation. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 211–220.

Apostolos Papadopoulos and Yannis Manolopoulos. 1997. Nearest Neighbor Queries in Shared-Nothing Environments. *Geoinformatica* 1, 4 (Dec. 1997), 369–392. DOI:http://dx.doi.org/10.1023/A:1009758427967

Apostolos N. Papadopoulos and Yannis Manolopoulos. 1998. Similarity Query Processing Using Disk Arrays. *SIGMOD Rec.* 27, 2 (June 1998), 225–236. DOI:http://dx.doi.org/10.1145/276305.276325

Deyuan Qiu, Stefan May, and Andreas Nuchter. 2009. GPU-accelerated nearest neighbor search for 3D registration. In *Computer Vision Systems*. Springer, 194–203.

Nikos Sismanis, Nikos Pitsianis, and Xiaobai Sun. 2012. Parallel search of k-nearest neighbors with synchronous operations. In *High Performance Extreme Computing (HPEC), 2012 IEEE Conference on*. IEEE, 1–6.

Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. 2008. Real-time KD-tree construction on graphics hardware. In *ACM Transactions on Graphics (TOG)*, Vol. 27. ACM, 126.