PCAF: Scalable, High Precision k-NN Search using Principal Component Analysis based Filtering

Huan Feng*, David Eyers†, Steven Mills†, Yongwei Wu*, Zhiyi Huang†
* Tsinghua University, China † University of Otago, New Zealand

***Abstract*—Approximate k Nearest Neighbours (AkNN) search is widely used in domains such as computer vision and machine learning. However, AkNN search in high dimensional datasets does not work well on multicore platforms. It scales poorly due to its large memory footprint. Current parallel AkNN search using space subdivision for filtering helps reduce the memory footprint, but leads to loss of precision. We propose a new data filtering method—PCAF—for parallel AkNN search based on principal components analysis. PCAF improves on previous methods by demonstrating sustained, high scalability for a wide range of high dimensional datasets on both Intel and AMD multicore platforms. Moreover, PCAF maintains high precision in terms of the AkNN search results.**

I. Introduction

Wide use of k Nearest Neighbours (k-NN) search is made in domains such as bioinformatics [6], data analysis [9], machine learning [37], computer vision [41] and handwriting recognition [42]. Given query data points, k-NN finds k data items within a database (*i.e.*, a set of features) that are most similar to the query data, where the similarity is often measured by Euclidean Distance. In general, a feature f can be defined as a D dimensional vector: $f = [e_1, e_2, .., e_D]$. The database DB is defined as a set of N features: $DB = \{f_1, f_2, .., f_N\}$. We call the feature that is used to query the database DB the *query feature* and the features in DB the *reference features*. Based on these definitions, the k-NN problem can be formally described as: given a query feature q, find the k reference features in DB that have the shortest (Euclidean) distances to q.

To address the challenge of rapidly increasing amounts of data being included for processing, many Approximate k Nearest Neighbours (AkNN) algorithms [5], [7], [21], [32], [34] have been proposed. Instead of returning the actual k-NN, they return k results that are highly likely to be the k-NN. Although AkNN algorithms have better performance, their searching precision is of great concern [17], [23], [36].

There are two main strategies in AkNN for finding approximate nearest neighbours: *data selection* and *data filtering*. The data selection strategy tries to find candidate features that are most likely to be the precise k nearest neighbours. Most AkNN algorithms adopt this strategy [1], [7], [29]. However, this strategy incurs a large memory footprint and the AkNN algorithms have poor scalability on multicore systems due to them performing a large number of random memory accesses and thus causing cache misses [8], [34], [35].

The data filtering strategy [35] instead excludes unlikely features based on distance estimation between the query feature and the reference feature. If they have a high filtering rate, much computation and many memory accesses can be avoided. Typically, the scalability of AkNN can be greatly improved on multicore systems by using a filtering strategy.

Subspace Clustering for filtering (SCF) [34] was the state-of-art approach using the data filtering strategy in AkNN. It greatly improves the scalability of AkNN algorithms. However, its search precision is unstable and depends on the nature of the reference features. We will discuss this challenge in detail in the next section.

In this paper, we propose a parallel AkNN algorithm called PCAF which uses principal Components Analysis (PCA) [18] to estimate the rank of distance between the query feature and the reference feature. PCAF uses data filtering to exclude those reference features that are not likely to be k-NN features according to the PCA estimation. It has high scalability on multicore systems with stable, high search precision on high-dimensional datasets (*e.g.*, 561 dimensions).

The remainder of this paper is organised as follows. Section II describes the motivation of the idea. Section III presents our PCAF algorithm. Section IV demonstrates the detailed technical implementation of PCAF. Section V provides experimental results and evaluation compared with four widely-used k-NN algorithms. Section VI discusses the most related work. Finally, Section VII summarises the contributions of this paper.

II. Motivation

As discussed previously, data filtering in AkNN greatly improves its parallel performance on multicore systems. We have previously proposed a parallel AkNN algorithm called SCF using data filtering to exclude unlikely k-NN features. Before searching, SCF needs to build an index for the reference features, as needed in all AkNN algorithms. SCF divides the reference features into a number of subspaces with low dimensionality in order to alleviate the problem of *curse of dimensionality* [4], [9]. Then in each subspace, SCF uses k-means [26], [29] clustering method to divide the reference features into clusters. The centre of each cluster is used to estimate the distance between the query feature and the reference features of the cluster in the subspace. Finally the distance between the query feature and any reference feature in

* Department of Computer Science and Technology; Tsinghua National Laboratory for Information Science and Technology (TNLIST) Tsinghua University, Beijing 100084; Technology Innovation Center at Yinzhou, Yangtze Delta Region Institute of Tsinghua University, Ningbo 315000, Zhejiang; Research Institute of Tsinghua University in Shenzhen, Shenzhen 518057, China
† Department of Computer Science, University of Otago, New Zealand

TABLE I: The rank estimation using SCF with different subspaces and PCAF in an example

	e_1	e_2	e_3	e_4	RED	Rank
q	1	1	1	1	-	-
A	1	1	14	15	19.10	4
B	2	3	7	11	11.87	2
C	4	5	5	5	7.55	1
D	5	6	12	10	14.90	3

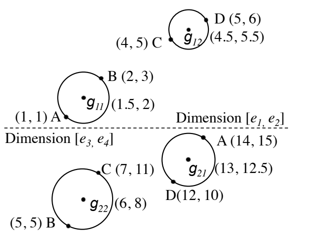
(a) A 4-dimensional case with one query feature, q, and four reference features, A, B, C, and D. RED and the exact rank of the reference features are listed.

SCF	$[e_1\ e_2]\ [e_3\ e_4]$		$[e_1\ e_3]\ [e_2\ e_4]$	
	EED	Rank	EED	Rank
A	16.66	3	17.12	4
B	8.67	1	13.19	2
C	10.32	2	9.57	1
D	17.57	4	14.51	3

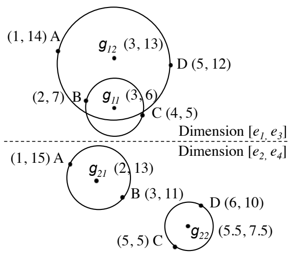
(b) EED and rank using SCF with a different formation of subspaces, sequential manner ($[e_1\ e_2][e_3\ e_4]$) and interleaved manner ($[e_1\ e_3][e_2\ e_4]$)

PCAF	p_1	EED	Rank
q	-10.77	-	-
A	7.29	18.06	4
B	-0.73	10.04	2
C	-7.05	3.72	1
D	0.48	11.25	3

(c) EED and rank using PCAF with only 1 principal component for projection. p_1 lists the projections of each feature



(a) Forming subspaces in a sequential manner, so that features in subspace $[e_1\ e_2]$ are divided into 2 groups with centres g_{11} and g_{12}, while features in subspace $[e_3\ e_4]$ are divided into 2 groups with centres g_{21} and g_{22}



(b) Forming subspaces in an interleaved manner, so that features in subspace $[e_1\ e_3]$ are divided into 2 groups with centres g_{11} and g_{12}, while features in subspace $[e_2\ e_4]$ are divided into 2 groups with centres g_{21} and g_{22}

Fig. 1: Using SCF to divide space into two subspaces with two groups, each formed in a different way

the original high-dimensional space is estimated by summing up the distance between the query feature and the reference feature in each subspace.

Table Ia gives an example of k-NN search with one query feature and four reference features in a 4-dimensional space. The Real Euclidean Distance (RED) and the rank based on it are also listed in the table. Though we use only one query feature and four reference features here, it is worth pointing out that in real applications such as image processing, there will typically be tens of thousands of query features and reference features in each pair of images and tens of thousands of images to be searched and matched pair by pair. This domain thus demands high-performance parallel computing if results are to be calculated rapidly.

We use the aforementioned example to demonstrate how SCF works. Suppose SCF divides the feature space into two subspaces in a sequential manner: the dimensions $[e_1, e_2]$ form one subspace and $[e_3, e_4]$ form the other. In each subspace, two clusters are formed and the centre of each cluster is used to estimate the distance between the query feature and the reference features of the cluster, as shown in Figure 1a. Based on the subspaces in Figure 1a, the Estimated Euclidean Distance (EED), which is calculated by summing up distances from the query to each group centres, is shown in Table Ib (for more details about how the EED is calculated refer to [34]).

From the table, we know the rank of the reference features is B, C, A, D according to the EED. However, according to the RED in Table Ia, the rank is C, B, D, A. Based on the estimation of SCF, if only one nearest neighbour is requested, in this case, the precision of the 1-NN result using SCF is 0—the matched feature is not actually the closest.

However, if we change the partitioning of dimensions for the subspaces as in Figure 1b, SCF can produce the correct 1-NN result. In the figure, we instead choose $[e_1, e_3]$ to form the first subspace and $[e_2, e_4]$ to form the second subspace. Likewise we create two clusters in each subspace. According to the EED calculated by SCF, the rank of the reference features is C, B, D, A which is exactly the same as their rank in RED. Therefore, the precision of each of k-NN for $k \in 1..4$ using SCF would be 100%.

From the above example, we can see that the precision of SCF is seriously affected by how the subspaces are formed. Forming the best subspaces to achieve the highest precision is data dependent, and difficult to determine in SCF [10], [34]—it depends on the nature of the reference features, which are different in different applications.

In the following section, we propose a PCA based filtering method, PCAF, for AkNN search. It has the same advantages of using the data filtering strategy: high scalability, small memory footprint, and reduced computational overhead. More importantly, compared with SCF, PCAF has stable, higher precision k-NN results due to its accurate rank estimation of Euclidean distance using the principal components of the reference features.

III. PCA-based Filtering (PCAF)

Principal Components Analysis (PCA) is a popular algorithm used to reduce dimensionality, that can alleviate the curse of dimensionality in some contexts. It uses an orthogonal transformation to convert a set of data values of possibly correlated variables into a set of data values of linearly uncorrelated variables called *principal components*.

For a high dimensional dataset, if the correlation between the dimensions is strong, the information of the dataset can be represented by a small number of principal components, which taken together have lower dimensionality. By using these principal components, the dataset can find a projection, which has the same dimensionality of principal components, containing most of its original information.

For example, the reference features in Table Ia can be represented by projection p_1, when using one principal component after PCA is applied to the set of four reference features, as shown in Table Ic. The space of the principal components is called the *projected space* of PCA.

PCAF uses the projections under principal components to estimate the rank of distances between the query feature and the reference features. Note that we use the projections to estimate the rank of distances instead of the real distances, since the distances calculated with projections are in a different, transformed low-dimensional space, which is different from the original, high-dimensional feature space. As the principal components are the main factors deciding the relationships among the reference features, the rank estimation of distances with corresponding projections should be similar to the rank of distances in the original feature space, if the dimension reduction has not lost too much data. That is, if the distance between the query feature and a reference feature is small in the original feature space, the distance of the feature from the query in the projected space of PCA should be proportionally small. From Table Ic, we can see that the rank of distances between the query feature and the reference features in the projected 1-dimensional space is C, B, D, A, which is exactly the same as the rank of those in the original feature space.

In many real-world domains, the feature dimensions are more or less correlated [28]. For example, SIFT features, which are popular in computer vision applications, have many dimensions that are correlated with each other [10], [38]. From our experiments, often we only need around 10 dimensions in the PCA space projected from spaces of hundreds of dimensions. This shows us that we could possibly use very low-dimensional PCA-converted features to accurately estimate the rank of distances between the query feature and the reference features. This approach can save a large amount of computation by calculating the distances of a very low-dimensional, surrogate space.

In PCAF, before searching, PCA is applied to the reference features to find the principal components denoted as the PCA space. That is, the reference features are projected into the surrogate PCA space. Then the query feature is projected into the same PCA space. PCAF maintains two heaps for the current k nearest neighbours. The main heap contains the current k nearest neighbours ranked by the distance of the original feature space. The assistant heap contains the current k nearest neighbours ranked by the distance of the PCA space.

During searching, the distance between a reference feature and the query feature in the PCA space is calculated first. If the distance is larger than the largest distance in the assistant heap, we simply drop the reference feature as it is unlikely to be one of the k nearest neighbours. In this way, we can filter out a large number of reference features without much computational overhead. However, if the distance is smaller than the largest distance in the assistant heap, PCAF calculates the real distance between the reference feature and the query feature in the original feature space, which has much higher computational overhead. If the distance is larger than the largest distance in the main heap, we drop the reference feature; otherwise, the distance of the reference feature is inserted into the main heap and the corresponding distance in the PCA space is inserted into the assistant heap. After each reference feature is processed as described above, the k nearest neighbours are in both heaps.

In PCAF, there are two overheads that are related to PCA. The first is the PCA transformation applied to the reference features, which projects the reference features into the PCA space. Since the projected features will be used by a quarter of billion queries from tens of thousands of images, this one-off overhead is negligible and equivalent to the overhead of building indices in other AkNN algorithms according to our experiments. Moreover, if necessary, this PCA process could be parallelised to further reduce the overhead, as many parallel PCA algorithms have been proposed already [22], [24], [43].

The second overhead is the projection of the query feature into the PCA space. It involves a multiplication between a $D \times d$ matrix and a vector of size D, where D is the dimensionality of the original feature space and d is the dimensionality of the PCA space. This one-off overhead is amortised by the distance computation against tens of thousands reference features as the same projected query feature will be reused by each of those reference features. Our experimental results will show this one-off overhead is negligible. It is worth noting that, if necessary, this overhead could be easily parallelised by multiple cores or the SIMD-based floating point unit *e.g.*, SSE, though it is a relatively small overhead.

The advantages of PCAF can be summarised below. First, in most cases, it replaces the distance calculations between features in a high-dimensional space with the calculations in a surrogate low-dimensional space. The computational overhead is substantially reduced. Second, the memory footprint is greatly reduced as only the low-dimensional projections are accessed most of time. This is extremely helpful for those multicore systems with limited memory bandwidth. Third, the precision of k-NN results is significantly improved compared to other AkNN algorithms due to the use of principal components for distance estimation.

IV. Implementation of PCAF

The main idea of PCAF is to use the distance rank in the surrogate PCA space to filter out the reference features that are unlikely to be k-NN. In this section, we will discuss: (a) the rank estimation in the PCA space, (b) the filtering algorithm in detail, and (c) our fine-grained data parallelism in PCAF.

Note that, like other AkNN algorithms, to reduce computational overhead, we use the following Squared Euclidean Distance (SED), instead of Euclidean Distance, to measure the distance between two features f_A and f_B in a D dimensional space in the rest of the paper:

$$SED(f_A, f_B) = |f_A - f_B| = \sum_{i=1}^{D} (f_A[i] - f_B[i])^2$$

A. Rank Estimation with PCA

In our implementation, we use Singular Value Decomposition (SVD) [11] to find the principal components, which is represented by a $D \times d$ matrix denoted as W, where D is the dimensionality of the feature space, and d is the dimensionality of the PCA space or the number of principal components, which is much smaller than D. Specifically, the two-sided Jacobi R-SVD decomposition provided by the Eigen library [12] is used to ensure optimal reliability and accuracy of PCA. With the principal components matrix W, for the reference feature set $R = (f_1, f_2, ..., f_N)$ or query feature q, we need to project everything into the PCA space using the following matrix-matrix or vector-matrix multiplication:

$$r_i = (f_i - f_{mean}) \qquad i = 0, 1, ..., N$$

$$R_{pca} = R_m \times W \qquad R_m = (r_1, r_2, ..., r_N)$$

$$q_{pca} = (q - f_{mean}) \times W$$

where f_{mean} is the mean of the reference features, which is available after PCA is done. This process can be accelerated through parallel computing since either parallelisation of SVD or matrix-vector structured multiplication has been well-studied [14], [22], [40].

As W and R_{pca} are used by all queries, the computation is one-off preprocessing. Although the complexity of this overhead is $O(D^3)$, other AkNN algorithms have similar overhead for building search indices. According to our tests, using a single core of the Intel Xeon processor in our experimental environment, this one-off time overhead of PCAF is between 0.01s and 4.5s, which can be further shortened if we only compute necessary components during the SVD decomposition process [13], while the index building overhead of other AkNN algorithms is between 0.01s and 70.5s depending on precision[1]. However, this overhead is negligible as it is one-off and used by tens of thousands queries.

After the projection, the distance in the PCA space between the query feature q_{pca} and each reference feature f_{pca} in set R_{pca} is calculated. This distance is used to rank the reference feature in the following filtering method.

B. Filtering Method

Algorithm 1 gives the detailed description of data filtering in PCAF. For each reference feature f, PCAF first calculates the distance between the projected query q_{pca} and the projected reference feature f_{pca}. If the distance $dist_{pca}$ is smaller than the maximum distance in $heap_{pca}$, the distance between q and f, $dist$, is calculated. If $dist$ is smaller than the maximum distance in $heap$, then the maximum is replaced by $dist$ in $heap$, and $dist_{pca}$ replaces the current maximum distance in $heap_{pca}$. The final k-NN results are in $heap$ after the above process is repeated for each reference feature. Note that, in Algorithm 1, m is used to adjust the size of $heap_{pca}$ to accommodate fluctuation of rank estimation. A value of 2 is enough for most cases, which incurs little overhead.

C. Data parallelism

PCAF is particularly suitable for parallel implementation since there is no dependence in the search of k-NN within each query. This is different from other AkNN algorithms which require sequential execution when the same query is retrieving the index of reference features.

[1]See code and raw data of experimental results available on GitHub. https://github.com/c30268056/PCAF

Algorithm 1: PCAF data filtering

```
Input: $R$: set of reference features
Input: $R_{pca}$: set of projected reference features
Input: $q$: query feature
Input: $q_{pca}$: query projection
Input: $k$: number of nearest features required
Input: $m$: heap size scaling
Output: $heap$: contains the $k$-NN results
1  Initialise max heap $heap$ with size $k$;
2  $heap.max \leftarrow \infty$ ;
3  Initialise temporary max heap $heap_{pca}$ with size $k \times m$;
4  $heap_{pca}.max \leftarrow \infty$ ;
5  for $f_{pca} \in R_{pca}$ do
6      $dist_{pca} \leftarrow SED(q_{pca}, f_{pca})$;
7      if $dist_{pca} < heap_{pca}.max$ then
8          $dist \leftarrow SED(q, f)$;
9          if $dist < heap.max$ then
10             $heap_{pca}.replace(dist_{pca})$;
11             $heap.replace(dist)$;
12 return $heap$ with the $k$ nearest features;
```

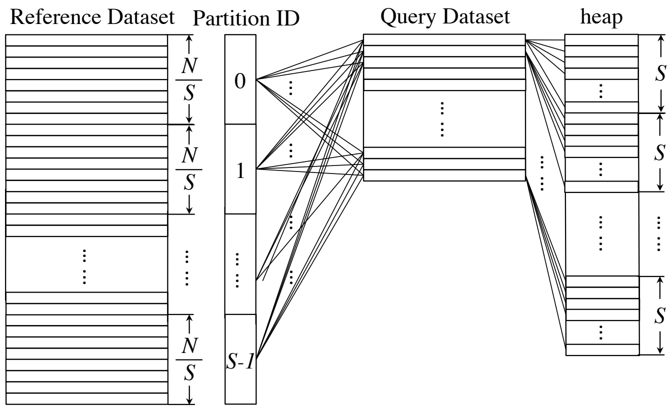


Fig. 2: Fine-grained parallel implementation based on data partition. Each task searches a set of $\frac{N}{S}$ features and maintains its own k-NN results in a heap.

In PCAF, the reference features in the PCA space are divided into S subsets which are searched in parallel by threads using the same query as shown in Figure 2. In the figure, each parallel processable task works on a subset of reference features and maintains its own heaps. As the size of the heaps is very small (2 for most image processing applications), the extra space overhead has no noticeable impact on the performance but the fine-grained parallelism supports high performance of PCAF.

After the k-NN results are obtained with each subset, the final k-NN results are computed using a simple selection algorithm among the results, like a k sized max heap structure for accumulation in our experiment.

D. Time and Space Complexity

We now analysing the complexity of our algorithm. PCAF takes little space to store the $d \times D$ transformation matrix, W, and D sized f_{mean} vector. During runtime, it will cost $O(Nd)$ space to store the projection of reference features, and $O(d)$ for each query projection. As the magnitude of N dominates the other parameters, the space complexity for PCAF can be simplified as $O(N)$.

PCAF can save time by reducing the number of SED calculations. Supposing the time for SED computing with all reference features for each query is denoted as T_D, and the filtering rate of excluded reference features is represented as FR, then theoretically PCAF takes $(1 - FR) \times T_D + \frac{d}{D} \times T_D$ time for SED computation. According to our experimental results, d is much smaller than D, and the FR reaches high above 95% for most cases. Besides, the extra cost for query projection during searching is only $O(dD)$. Thus, PCAF is very efficient by reducing many unnecessary distance computations.

We will illustrate detailed performance comparison with other k-NN algorithms in section V. Also note that by adjusting d, m (heap scale) and S (the number of data subsets) in PCAF, the performance and precision changes. Usually when d, m and S increase, the rank estimation accuracy improves with higher cost of time.

V. Evaluation

In this section, we evaluate the performance of our method against a brute-force k-NN algorithm, two state-of-art data selection AkNN algorithms and a data filtering AkNN algorithm on six real-world and synthetic datasets. The performance improvement that they attain on two multicore platforms is analysed.

A. Experimental Setup

1) Multicore platforms: Two multicore platforms are used in our evaluations:

a) Intel16: Intel(R) Xeon(R) CPU E5-2665, 8 cores × 2 sockets @ 2.40GHz, 20 MiB L3 shared cache, 128GiB DDR3 (1600 MHz) memory;
b) AMD64: AMD Opteron Processor 6276, 16 cores × 4 sockets @ 2.3 GHz, 16 MiB L3 shared cache, 512GiB DDR3 (1600 MHz) memory;

The Intel16 and AMD64 are two typical multicore platforms. The icc-14.0 compiler is used on Intel16 platform while gcc-4.8 is used on the AMD64 platform.

2) Algorithms: We compare our proposed algorithm with the four algorithms listed below:

a) Brute-force (BF): searches k-NN exhaustively in the whole database and gives the accurate results of k-NN.
b) Randomized kd-Trees (RKD): an efficient variant of the popular kd-tree [29] algorithm. Multiple trees are built as its index structure. During searching, it traverses these kd-trees and puts promising candidate nodes in a queue for distance calculation in the next step. The k-NN results searching within these nodes are considered to be the approximate k-NN results for the dataset.
c) Random Ball Cover (RBC): the state-of-art scalable k-NN algorithm on multicore platforms [7]. First, it randomly chooses several representative features to represent a number of reference subsets with size s, each of which contains s reference features that are nearest to one representative. When searching for approximate k-NN results, it finds the nearest representatives and then searches k-NN within those subsets of the representatives using BF.
d) Subspace Clustering for Filtering (SCF): the latest algorithm which implements data filtering strategy into k-NN search problem [34], as discussed before.

We implement BF and RKD by using the FLANN library [30], which provides fast approximate k-NN search functionality for computer vision related tasks. The implementations of RBC and SCF are taken from the open sourced code mentioned in previous work. The parallelisation is carried out by using OpenMP.

3) Datasets: The datasets listed in Table II are used to evaluate the performance of the aforementioned algorithms. Though practical query datasets are much larger, we use small query datasets to save the time of experimental runs, which does not affect the validity of our results. As a matter of fact, the benefit of PCAF will be amplified using larger query datasets since PCAF reduces computational and memory costs for each query.

TABLE II: Overview of test datasets

Name	Size (MiB)	Dimen-sionality	Number of reference features	Number of query features
Digits	0.93	64	3823	1797
Random	12.21	128	25000	7500
SIFT	27.38	128	56074	9929
Madelon	3.81	500	2000	1800
GIST	5.86	512	3000	1000
HAR	15.73	561	7352	2947

a) “Digits” is the smallest dataset in both size and dimensionality. It is a real-world dataset [3] that contains handwritten digits that have been size-normalised and centred into a fixed 8×8 image.
b) “Random” is a synthetic dataset, which contains features that are chosen from a uniform distribution.
c) “Madelon” is an artificial dataset [15] containing data points grouped in 32 clusters placed on the vertices of a five dimensional hypercube.
d) “SIFT” is the largest dataset and generated by extracting SIFT descriptors [25] from real images. SIFT is one of the most widely-used image feature detectors in computer vision [20].
e) “GIST” contains features collected from 4000 images by using GIST descriptors [33]. GIST summarises the gradient information of different parts of an image and converts them into one high dimensional feature. It is also a frequently used algorithm in computer vision.
f) “HAR” [2] has the highest dimensionality of the datasets that we use. It contains sensor readings related to 30 real-world subjects performing activities of daily living,

which are recorded by the embedded sensors in a waist-mounted smart phone.

4) Parameter Settings: In order to find the performance bounds of each algorithm, we conduct as many scenarios as possible by adjusting parameters. For simplicity, we denote a parameter $X \in [start..end, \Delta step]$ as the value of X is set from $start$ to end with an increase of $step$.

The RKD algorithm has two important parameters. The number of trees to build for indices greatly affects both indexing time and searching precision. The checks parameter represents the number of neighbouring buckets that should be checked during the search. As these parameters are increased, search time increases, but the results will be more precise [29]. We set trees as 4, 8, 16, 32, 64 or 128 and tested checks $\in [128..5120, \Delta 128]$.

The number of randomly chosen representatives in RBC positively affects the precision and negatively affects the searching time [7]. Suppose $n = \lfloor \sqrt{N} \rfloor$ where N is the number of the reference features, then the number of representatives is chosen from $[n..N, \Delta n]$.

For SCF, the number of subspaces and clusters are important parameters [34] and are set within $[4..64, \Delta 4]$ and $[8..32, \Delta 8]$ respectively.

The bounds of d in PCAF is decided by the percentage of information/variance in the reference dataset that is retained in PCA space. The lower and upper bound of d are set using the case when 50% and 90% of the variance is retained, respectively. The heap scale is set from 1 to 5, while reference features are partitioned into either 1, 2, 4, 8, 16 or 32 parts.

5) Evaluation Metrics: We consider three common metrics to evaluate the parallel performance, and we also use two more metrics to evaluate the filtering effectiveness of each of the data filtering algorithms:

a) Time: the total time used for searching.
b) Improvement: defined as the searching time of the exact (using BF) solution divided by searching time after applying an AkNN algorithm.
c) Speedup: defined as sequential searching time of an algorithm divided by its parallel searching time. It is used to measure the scalability of each algorithm.
d) Precision: defined as the percentage of k-NN results that are correctly found.
e) Filtering Rate: represents the percentage of reference features that are excluded. It is used to help measure the effectiveness of filtering method.

B. Comparison with Brute-force

Brute-force (BF) is the traditional and straightforward implementation of a k-NN algorithm that provides accurate results. PCAF shows a great enhancement over BF in both speed improvement and scalability when exact results (100% precision) are retrieved.

1) Improvement: In this section, we evaluate the parallel performance improvement when using all available cores. The improvements that PCAF can achieve over BF on various datasets are listed in Table III.

TABLE III: Performance improvement compared with BF runtime after applying PCAF to each dataset on each platform using all available cores without losing precision

Platform	Digits	Random	SIFT	Madelon	GIST	HAR
Intel16	2.47	1.07	3.66	1.99	5.85	2.25
AMD64	2.54	3.51	10.29	1.95	6.76	7.95

TABLE IV: Filtering Rate (FR) and dimensionality in PCA space (d) for PCAF to produce exact k-NN results for each dataset

	Digits	Random	SIFT	Madelon	GIST	HAR
$FR(\%)$	95.27	94.70	98.60	87.53	96.81	62.93
d	5	90	15	53	8	24

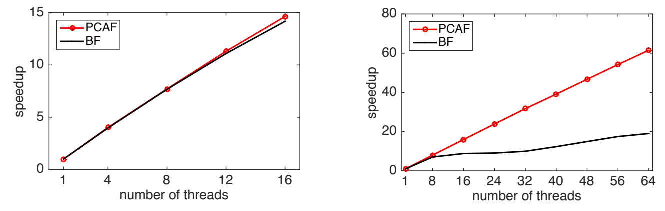
The improvement mainly comes from the saved distance computation. As the distance computation complexity of BF is $O(N \times D)$, however in PCAF, it is $O((1-FR) \times N \times D) + O(N \times d)$, where FR is the filtering rate of excluded reference features for distance computation, and d is the dimensionality in PCA space. From Table IV, we can observe a high FR and a small d in most of the datasets, which results in a large amount of computation reduction by PCAF.

The improvement varies from different datasets because the amount of computation reduction varies. Take the datasets "SIFT" and "Random", which have the same dimensionality of 128, as the example. The dimensionality greatly reduced in "SIFT": a 15-dimensional PCA space is enough to present most of the information from the original 128-dimensional features. However for "Random" dataset, 90 principal components have to be used for projections. As the filtering rates between "SIFT" (reaching 98.60%) and "Random" (94.70%) are close to each other, the searching time is highly affected by the cost of the rank estimation, which is based on SED distances of projections.

2) Scalability: PCAF has outstanding scalability for very high-precision k-NN search on multicore platforms. Compared with BF, PCAF only frequently accesses $\frac{d}{D}$ of the memory during computation, which requires significantly fewer memory accesses—d is much smaller than D. Due to space restrictions we only use the largest dataset ("SIFT") as the example here to compare the scalability with BF for highly precise (in this case exact) k-NN searching.

As shown in Figure 3a, both PCAF and BF have considerable scalability on Intel platforms. Note that from examining the hardware performance monitoring counters, PCAF has a 20% reduction in L2 cache misses compared to BF, but the scalability improvement due to fewer memory accesses is minor. This is because the Intel platform has a high memory bandwidth, so that the memory wall problem is not obvious. However, as the trend of parallel computing development engages more and more cores, the memory bandwidth will eventually become an unavoidable issue.

Figure 3b shows the scalability on AMD platform, where PCAF provides significant scalability while the curve of BF is quite flat. This is because unlike the Intel platform, the 64-core AMD platform suffers serious memory latency issues. The statistics from performance monitoring counters shown in



(a) Intel16 (b) AMD64

Fig. 3: Scalability of PCAF and BF for "SIFT" dataset to find exact k-NN results on Intel16 and AMD64 platforms

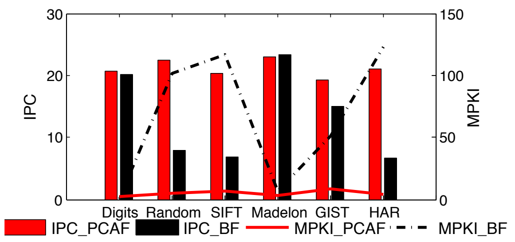


Fig. 4: Performance monitoring counter statistics on the AMD64 platform

Figure 4 help explain the results: the improvement of retired Instructions Per Cycles (IPC), which indicates the computation efficiency of the algorithm, is highly related to the reduction of last-level cache Misses Per (1000) Instructions (MPKI). That is when MPKI decreases significantly, IPC increases accordingly which leads to better scalability. Since the MPKI of PCAF is really small in all the cases, PCAF can be seen to be cache-friendly, and thus achieves substantial scalability.

C. *Compared with Data Selection Algorithms*

Randomized kd-Trees (RKD) and Random Ball Cover (RBC) are two typical *data selection* algorithms for AkNN search. RKD is highly efficient but its tree-based index structure becomes a barrier to achieving high parallel performance. RBC is developed as a state-of-art k-NN algorithm for multicore platforms, however it involves a large amount of unnecessary distance computations.

Compared with RKD and RBC where approximate k-NN results are required, PCAF produces higher precision results within shorter searching times, and shows large improvements in scalability.

1) Improvement: Figure 5 shows the searching time of each algorithm on the Intel16 platform when using all available cores for different datasets reaching above 90% precision. We also marked the searching time of BF in the figure for reference. Due to space constraints, we don't present the AMD64 results as they show the same general pattern as the Intel16 platform.

As we can see from Figure 5, PCAF is the only method that can provide exact results for every dataset. It is also the quickest to produce high precision k-NN results. As the majority of unnecessary distance computation of between original features is filtered, the advantage becomes more obvious when the dimensionality of dataset increases.

RKD produces k-NN results with good precision in a very short time only for lower-dimensional datasets such as "Digits". This is because RKD divides the reference space into bins along the axis of dimensionality, which makes it efficient for searching in low dimensional space [29]. When it comes to high-dimensional space where the *curse of dimensionality* arises, RKD needs to spend much more time on visiting many more branches to achieve high precision [31]. Moreover, the precision of k-NN results found by RKD converges at a certain precision ($< 100\%$).

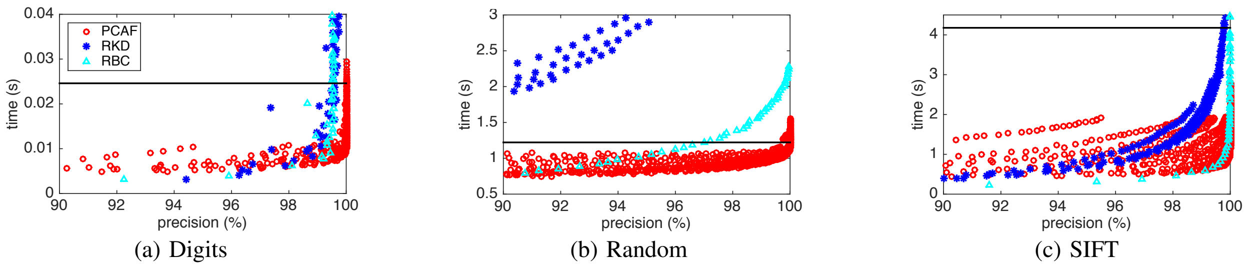
Apparently RBC performs better than RKD in most of the cases but it still produces high precision results more slowly than PCAF. The only exceptions to this occur for lower precision (<99%) k-NN results from the "Digits" and "SIFT" datasets, where RBC is slightly faster than PCAF. However RBC cannot sustain results at high precision. Above 99.55% ("Digits") and 99.99% ("SIFT") respectively, RBC takes $5.18\times$ and $2.91\times$ the searching time of PCAF.

The improvement mainly comes from computation being avoided in PCAF, as shown in Figure 6a. Based on performance monitoring counters, in the figure, the ratio of total retired instructions compared with BF in PCAF is the lowest. The large proportion of computation avoided, compared to RKD and RBC brings PCAF the substantial improvement observed in the experiment. Note that the reason why the number of retired instructions in RBC is more than that in BF is that RBC requires redundant distance computation to achieve high precision.

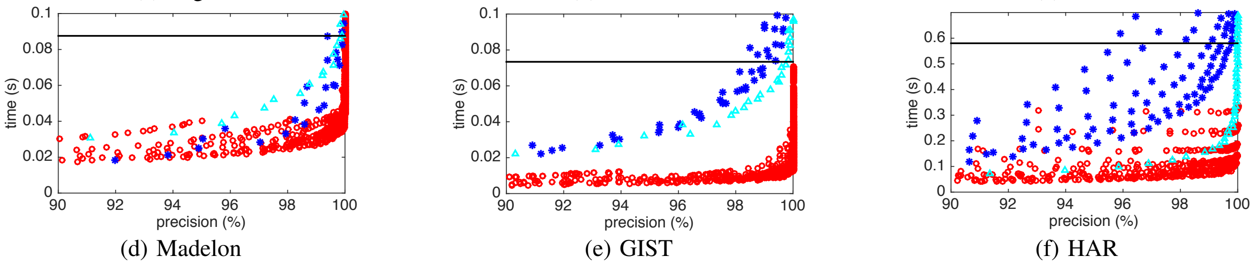
2) Scalability: In this section, we evaluate the scalability of our algorithm compared with RKD and RBC. Given the space available, we only choose "Madelon", "SIFT" and "HAR" datasets as examples. As similar characteristics and scalability patterns are shared across all datasets, these three are sufficient to be representative. "Madelon" is quite a small dataset, as are "Digits" and "GIST". "HAR" is much larger than "Madelon", and the dimensionality is the highest of all the datasets. "SIFT" is the biggest dataset in our experiments, but the dimensionality is only a quarter of "HAR", and "Random" is similar. To consider the scalability of each algorithm under a reasonable search precision, we plot the average speedup of selected cases that produce results that are more than 95% accurate. We also excluded results that have longer search times than brute force.

The difference of scalability between each algorithm running on Intel platform is not significant shown in Figure 7. But we can still observe an vivid inferior rank of either RBC or RKD, especially when dataset becomes larger. (The reason has been explained in Section V-B2.)

Unlike the Intel platform, the AMD platform showed significant variance within repeated runs of our experiments. We thus add the maximum speedup that we observed in each case as a dotted line on the graphs in Figure 8. As "Madelon" is small enough to fit in the last-level cache, similar to the Intel platform, the scalability improvement is not obvious. However in the case of "SIFT" and "HAR", a disparity is seen. In all cases, PCAF reaches the best maximum scalability, and

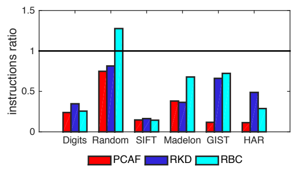


(a) Digits (b) Random (c) SIFT

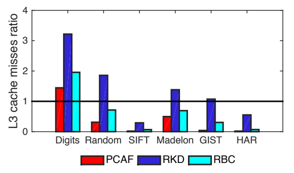


(d) Madelon (e) GIST (f) HAR

Fig. 5: Searching time under different precision ($> 90\%$) for each dataset on the Intel16 platform using all available cores. Note that the black line shows the time cost for BF. The legend in (a) also applies to (b)–(f).



(a) Computation reduction



(b) Memory reduction

Fig. 6: Ratio of performance monitoring counters compared with BF for each algorithm on AMD64. Bars below 1.0 present desirable reductions, otherwise show the undesired overhead.

provides the best average speedup.

The last level cache misses recorded in performance monitor counters indicates the size of memory footprint. From the ratio of misses compared with BF shown in Figure 6b, PCAF can be seen to gain memory performance in almost all cases except "Digits", which is a small dataset. For very small datasets like "Digits", the benefit of reduced memory footprint is overshadowed by the non-contiguous, interleaved memory accesses to features and PCA-projected features, though the benefit of reduced computation still dominates the overall performance of PCAF.

D. Compared with Data Filtering Algorithms

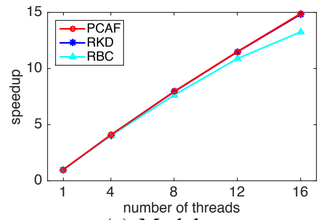
As far as we know, SCF is the only other published approach to AkNN searching that adopts a *data filtering* strategy. Compared to SCF, PCAF produces stable and higher precision results with less searching time, while exhibiting similar scalability. In this section, we evaluate both the parallel performance and the effectiveness of the filtering.

The frequently accessed index in SCF has an $O(NS)$ space complexity, where S is the number of subspaces (usually $S < 64$) (for detailed analysis refers to [34]). While PCAF spends $O(Nd)$ to store the counterparts—projected reference set denoted as R_{pca}. Supposing each element in R_{pca} is a 32-bit floating point number in PCAF, while the index for each feature in SCF requires one byte of storage, then in the cases when $d < \frac{S}{4}$, PCAF will have a smaller memory footprint. As the dimensionality of projections in PCA space depends on the properties of the dataset, the d in our implementation for "Random" and "Madelon" is quite large (when 50% of the variance is retained, d is 61 and 53 respectively), while for other datasets, d is relative small —less than the dimensionality needed in the case of exact searching shown in Table IV. Still, the difference between d and $\frac{S}{4}$ is not significant, which results in comparative scalability. The speedup curves shown in Figure 9 support our analysis. As we can see, PCAF is only slightly better in scalability compared with SCF.

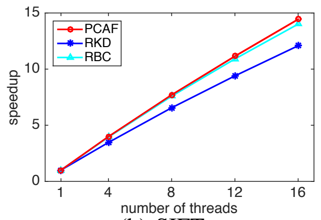
However, while achieving comparable performance in terms of scalability, PCAF greatly improves the speed for high precision AkNN searching. This is because the filtering strategy of PCAF is more effective than SCF's, so that PCAF has a higher FR in most cases, and thus avoids computation more effectively. Likewise, when achieving the same filtering rate, PCAF produces more precise results. As for a data filtering algorithm, the k-NN results found come from the non-filtered features, thus we compare the filtering effectiveness by plotting the precision achieved under the same filtering level shown in Figure 10. Due to limited space, we only present the high FR levels between the range of $(95, 97]$ and $(97, 100]$, which are also key regions of concern for data filtering AkNN algorithms. Also note that the blank in "Madelon" for SCF is because the highest FR SCF can reach for "Madelon" is only 82.74%. As we can see, PCAF produces nearly 100% precise results when $FR > 95$, which is higher than SCF in all datasets. Thus there is a higher probability of the PCAF filtering method discovering the true k-NN features during searching, while false filtering occurs in SCF.

VI. Related Work

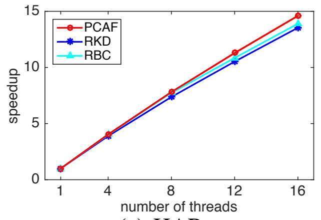
The focus of our work is speeding up high dimensional, high precision k-NN search on multicore architectures. We



(a) Madelon

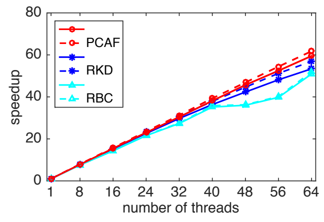


(b) SIFT

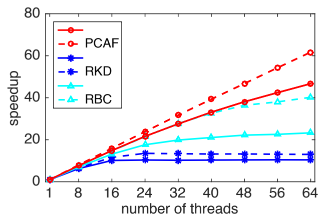


(c) HAR

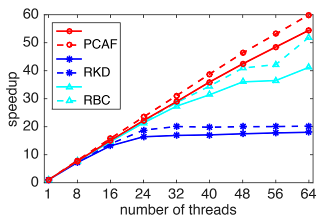
Fig. 7: Scalability of each algorithm on the Intel16 platform, for precision above 95%. The y-axis represents the speedup over sequential algorithm.



(a) Madelon

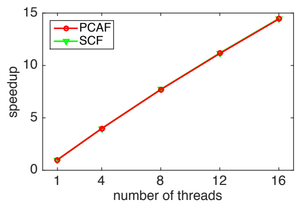


(b) SIFT

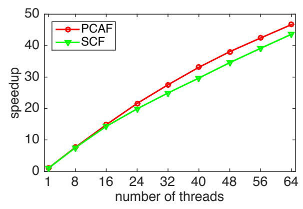


(c) HAR

Fig. 8: Scalability of each algorithm on the AMD64 platform, for results with a precision above 95%. The y-axis represents the speedup over the sequential algorithm. The dotted line represent the maximum speedup that was observed.

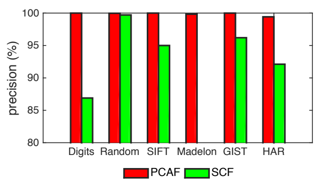


(a) Intel16

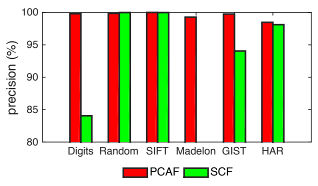


(b) AMD64

Fig. 9: Scalability of SCF and PCAF for "SIFT" dataset on Intel16 and AMD64 platforms.



(a) FR between (95,97]



(b) FR between (97,100]

Fig. 10: Precision of PCAF and SCF for the same filtering level

are not aware of many similar efforts to optimise AkNN algorithms for performance and precision in this way. Still, the idea behind our method stems from related research.

The algorithm previously proposed by Tang [34], [35] is the key work defining the notion of data filtering for AkNN. The rationale is to use a low-cost method to filter away many expensive distance computations, and maintain a small memory footprint at the same time to achieve high parallel performance. We adopt the notion of data filtering in our work. However, the precision of Tang's SCF algorithm is not stable, and is highly dependent on the nature of the dataset used.

Many real-world high dimensional datasets are actually governed by a small number of dominant dimensions, as discussed in [10] which inspires our data filtering method using lower dimensional projections. The idea of using low-dimensional intrinsic structure in high dimensional space is popular in computer vision [4], [16], [27]. Many solutions proposed for k-NN search in computer vision use dimensionality reduction techniques to either form lower dimensional feature descriptors [19] or sort features in advance [39]. However, these techniques either reduce the dimensionality of the feature space directly which results in loss of feature information, or incur extra overhead *e.g.* sorting, while in PCAF we use dimensionality reduction for filtering process without changing the original feature space for searching, which retains high precision of k-NN search and reduces computation.

VII. Conclusions

Data selection AkNN algorithms have serious memory bottlenecks on multicore systems. Using a data filtering strategy that reduces computation and memory footprint can improve scalability. However it can be challenging to maintain the accuracy of the result set with reduced computation. In this paper, a novel filtering method, PCAF, is proposed to exclude unlikely k-NN features with accurate estimation of similarity in high-dimensional space. Experimental results show that PCAF achieves substantial speedups and good scalability on multicore platforms compared with many data selection AkNN algorithms, and provides higher precision than an existing data filtering algorithm.

VIII. Acknowledgements

We thank the anonymous reviewers for their valuable comments. Huan Feng would like to thank the University of Otago and China Scholarship Council for offering and sponsoring her internship during the course of this research. We are grateful to Xiaoxin Tang for his generous provision of source code and support for the evaluation of the SCF algorithm. This work is supported by Natural Science Foundation of China (61433008, 61373145, 61170210, U1435216), the National Basic Research (973) Program of China (2014CB340402), National High-Tech R&D (863) Program of China (2013AA01A213), and the Chinese Special Project of Science and Technology (2013zx01039-002-002).

REFERENCES

[1] M. Al Hasan, H. Yildirim, and A. Chakraborty. SONNET: Efficient approximate nearest neighbor using multi-core. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 719–724. IEEE, 2010.

[2] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Ambient assisted living and home care*, pages 216–223. Springer, 2012.

[3] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Database Theory ICDT99*, pages 217–235. Springer, 1999.

[5] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, pages 97–104. ACM, 2006.

[6] J. Buhler. Provably sensitive indexing strategies for biosequence similarity search. *Journal of Computational Biology*, 10(3-4):399–417, 2003.

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

[8] G. Cong and K. Makarychev. Optimizing large-scale graph analysis on multi-threaded, multicore platforms. In *Parallel & Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International*, pages 414–425. IEEE, 2012.

[9] D. L. Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 2000.

[10] H. Feng, S. Mills, D. Eyers, X. Shen, and Z. Huang. Optimal space subdivision for parallel approximate nearest neighbour determination. In *Proceedings of 30th Internation Conference on Image and Vision New Zealand*. IEEE, 2015.

[11] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numerische mathematik*, 14(5):403–420, 1970.

[12] G. Guennebaud, B. Jacob, et al. Eigen: a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms, 2012.

[13] G. Guennebaud, B. Jacob, et al. Eigen v3.2.8 JacobiSVD class template reference. http://eigen.tuxfamily.org/dox/classEigen_1_1JacobiSVD.html/, 2016.

[14] J. A. Gunnels, G. M. Henry, and R. A. Van De Geijn. A family of high-performance matrix multiplication algorithms. In *Computational Science, ICCS 2001*, pages 51–60. Springer, 2001.

[15] I. Guyon, S. Gunn, A. Ben-Hur, and G. Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems*, pages 545–552, 2004.

[16] G. Hua, M. Brown, and S. Winder. Discriminant embedding for local image descriptors. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[17] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 33(1):117–128, 2011.

[18] I. T. Jolliffe. *Principal Components Analysis, Second Edition*. Springer, 2002.

[19] Y. Ke and R. Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–506. IEEE, 2004.

[20] N. Khan, B. McCane, and S. Mills. Better than SIFT? *Machine Vision and Applications*, 26(6):819–836, 2015.

[21] J. M. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of Computing*, pages 599–608. ACM, 1997.

[22] S. Lahabar and P. Narayanan. Singular value decomposition on GPU using CUDA. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–10. IEEE, 2009.

[23] T. Liu, A. W. Moore, K. Yang, and A. G. Gray. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems*, pages 825–832, 2004.

[24] W. Liu, H. Zhang, D. Tao, Y. Wang, and K. Lu. Large-scale paralleled sparse principal component analysis. *Multimedia Tools and Applications*, pages 1–13, 2014.

[25] D. G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[26] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on Mathematical Statistics and Probability*, pages 281–297. Oakland, CA, USA., 1967.

[27] K. Mikolajczyk and J. Matas. Improving descriptors for fast tree matching by optimal linear projection. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[28] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.

[29] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2:331–340, 2009.

[30] M. Muja and D. G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(11):2227–2240, 2014.

[31] S. A. Nene and S. K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(9):989–1003, 1997.

[32] S. O'Hara and B. A. Draper. Are you using the right approximate nearest neighbor algorithm? In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 9–14. IEEE, 2013.

[33] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[34] X. Tang, Z. Huang, D. Eyers, S. Mills, and M. Guo. Scalable multicore k-NN search via Subspace Clustering for Filtering. *Parallel and Distributed Systems, TPDS, IEEE Transactions on*, 26(12):3449–3460, 2015.

[35] X. Tang, S. Mills, D. Eyers, K.-C. Leung, Z. Huang, and M. Guo. Data filtering for scalable high-dimensional k-NN search on multicore systems. In *Proceedings of the 23rd international symposium on High-performance Parallel and Distributed Computing, HPDC 2014*, pages 305–310. ACM, 2014.

[36] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 563–576. ACM, 2009.

[37] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[38] G. Treen and A. Whitehead. Efficient SIFT matching from keypoint descriptor properties. In *Applications of Computer Vision (WACV), 2009 Workshop on*, pages 1–7. IEEE, 2009.

[39] G. Treen and A. Whitehead. A PCA-based binning approach for matching to large SIFT database. In *Computer and Robot Vision (CRV), 2010 Canadian Conference on*, pages 9–16. IEEE, 2010.

[40] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix–vector multiplication on emerging multicore platforms. *Parallel Computing*, 35(3):178–194, 2009.

[41] Z. Wu, Q. Ke, M. Isard, and J. Sun. Bundling features for large scale partial-duplicate web image search. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009*, pages 25–32. IEEE, 2009.

[42] C. Zanchettin, B. L. D. Bezerra, and W. W. Azevedo. A KNN-SVM hybrid model for cursive handwriting recognition. In *Neural Networks (IJCNN), The 2012 International Joint Conference*, pages 1–8. IEEE, 2012.

[43] J. Zhang and K. H. Lim. Implmentation of a covariance-based principal component analysis algorithm for hyperspectral imaging applications with multi-threading in both CPU and GPU. In *2012 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*, pages 4264–4266. IEEE, 2012.