Scalable Multicore k-NN Search via Subspace Clustering for Filtering

Xiaoxin Tang, Zhiyi Huang, *Member, IEEE*, David Eyers, *Member, IEEE*, Steven Mills, and Minyi Guo, *Senior Member, IEEE*

Abstract—k Nearest Neighbors (k-NN) search is a widely used category of algorithms with applications in domains such as computer vision and machine learning. Despite the desire to process increasing amounts of high-dimensional data within these domains, k-NN algorithms scale poorly on multicore systems because they hit a memory wall. In this paper, we propose a novel data filtering strategy for k-NN search algorithms on multicore platforms. By excluding unlikely features during the k-NN search process, this strategy can reduce the amount of computation required as well as the memory footprint. It is complementary to the data selection strategies used in other state-of-the-art k-NN algorithms. A Subspace Clustering for Filtering (SCF) method is proposed to implement the data filtering strategy. Experimental results on four k-NN algorithms show that SCF can significantly improve their performance on three modern multicore platforms with only a small loss of search precision.

Index Terms—*k* Nearest neighbors, high-dimensional space, memory wall, multicore systems, scalability, subspace clustering for filtering

1 Introduction

Similarity search is very effective in solving statistical classification tasks from diverse domains such as computer vision [1], bioinformatics [2], data analysis [3], and handwriting recognition [4]. By finding similar items within a database of known items, existing knowledge can be used to predict unknown information. A frequently used class of algorithms for solving similarity search is k Nearest Neighbors (k-NN) search. Given any query object, the task is to find k data items within the database that are most similar to the query object, where the similarity is often measured in terms of euclidean distance.

With the rapidly increasing amount of data as we are entering the age of *big data* [5], efficient parallel algorithms for k-NN search are needed to make best use of multicore. Consider the vast volume of image data now available due to widespread digital photography, which makes computer vision a very interesting and challenging research field. In 3D reconstruction tasks, for example, Agarwal et al. [6] estimate that pairwise matching—which involves similarity search—on a dataset of 100,000 images using 500 processor cores would take 11.5 days. They avoid computing matches for all image pairs, but their final solution still spends 13 out of 21 hours computing feature matches when reconstructing a scene from 150,000 images on a cluster with 498 processor cores.

- *X. Tang and M. Guo are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Room 3-415 SEIEE building, No. 800 Dongchuan Road, Shanghai 200240, China. E-mail: tang.xiaoxin@sjtu.edu.cn, guo-my@cs.sjtu.edu.cn.*
- *Z. Huang, D. Eyers, and S. Mills are with the Department of Computer Science, University of Otago, PO Box 56, Dunedin 9054, New Zealand. E-mail: {hzy, dme, steven}@cs.otago.ac.nz.*

Manuscript received 10 June 2014; revised 4 Nov. 2014; accepted 7 Nov. 2014. Date of publication 19 Nov. 2014; date of current version 6 Nov. 2015. Recommended for acceptance by S. Guo. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TPDS.2014.2372755

In many image-processing systems such as those that match content from different images, features are first extracted from image files using algorithms like SIFT [7]. Then, similar features between images are sought using k-NN algorithms to find matched features between different images. Finally, these matched features can be used to locate common objects in the images so that they are recognized by the system.

In general, a feature f can be defined as a D dimensional vector: $f = [e_1, e_2, \ldots, e_D]$. The database X is defined as a set of N features: $X = \{f_1, f_2, \ldots, f_N\}$. We call the feature that is used to query the database X the "query feature" and the features in X 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 X that have the shortest (euclidean) distances to q. As image-processing applications are becoming more and more popular, the size of typical feature sets X is increasing. The dimensionality of features is also high: e.g., SIFT features have 128 dimensions.

Many approximate algorithms [8], [9], [10] have been proposed to deal with large sets of feature vectors. Instead of returning the actual k-NN, they return k results that are highly likely to be the k-NN. Though the precision of k-NN algorithms is very important, in many domains it is not necessary to be 100 percent accurate. In image processing, for example, the features themselves are only an approximate representation of the underlying data. By trading precision for performance, approximate algorithms can greatly improve the efficiency of k-NN search.

However, these approximate algorithms do not work efficiently on multicore systems [11], [12] due to memory latency and bandwidth issues (also known as the *memory wall*). In general, most approximate algorithms need two types of data structures: one for index data, and another for feature data. Both of these are frequently visited during k-NN search. The index structure is used for finding reference

features—called candidate features—that are most likely to be within the k-NN. To decide whether a candidate feature is one of the k-NN features, its distance to the query feature will be calculated using the feature data.

The index structure in approximate algorithms usually incurs random memory accesses that lead to many cache misses. The Randomized kd-tree (RKD) algorithm [10], for example, often visits different branches within its tree structure unpredictably during the search process. The data structure for the reference features is a matrix and can so requires O(ND) space. When both N and D are very large, which is often the case of high-dimensional problems, it can consume quite a lot of memory. For example, this structure can consume tens or hundreds of megabytes for a single image in the image matching problem. New algorithms, such as SONNET [11] and RBC [12], focus on designing cache-friendly and easy-to-parallelize index structures that have a more regular memory access pattern than earlier techniques. Unfortunately, they still require a great deal of unnecessary computation.

After carefully analyzing existing approximate k-NN algorithms, we have observed that they all use a *data selection* strategy. That is, based on certain characteristics of the feature space, the algorithms try to find some candidates that are most likely to be within the k-NN features. These algorithms work well in low-dimensional spaces. However, due to the problem that is dubbed the *curse of dimensionality* [3], [13], these techniques become rapidly less efficient as the dimensionality of the problem increases. Therefore, it takes more time, since they have to search more candidates, in order to maintain a reasonable search precision.

In this paper, we propose a novel *data filtering* strategy for high-dimensional k-NN search. Instead of finding the likely candidates in the data selection strategy, our data filtering strategy excludes those unlikely features based on distance estimation. The data filtering strategy has two advantages. First, it reduces computation and memory accesses by replacing high-dimensional distance calculation with less expensive distance estimation. Second, its index structure for filtering has a very small memory footprint and thus reduces the effect of the memory wall. Based on our experimental evaluation on three modern multicore platforms, we demonstrate that by combining the data filtering and data selection strategy, promising results can be achieved for k-NN search.

The contributions of this paper are as follows:

- We propose a novel data filtering strategy for k-NN search algorithms on multicore platforms. Its key idea is to reduce unnecessary computation and memory footprint of k-NN algorithms so that they can scale better on parallel platforms.
- We implement a Subspace Clustering for Filtering (SCF) algorithm to support efficient data filtering, which is achieved through accurate distance estimation based on clustering in multiple subspaces. Its structure is small enough to fit in the last-level cache, which reduces the effect of memory wall on multicore platforms.
- We develop a Reward-and-Penalty method to enable SCF to maintain a high filtering precision across different datasets. By adjusting the values of reward and penalty factors, SCF can help k-NN algorithms achieve higher performance with a small loss of search precision.

This paper is organized as follows: Section 2 presents our SCF method. Section 3 shows the experimental results of SCF. Section 4 discusses the tradeoff between performance and precision of SCF. Section 5 discusses the related work. Finally, Section 6 concludes this paper.

2 Data Filtering Strategy

In this section, we give the details of our data filtering strategy and how it is implemented. The following squared euclidean distance (SED) is used to measure the similarity between two features:

$$SED(f_i, f_j) = \|f_i - f_j\| = \sum_{m=1}^{D} \big(f_i[m] - f_j[m]\big)^2. \quad (1)$$

The square root in ED is not used in SED, which can reduce the computation without changing the ranking of results.

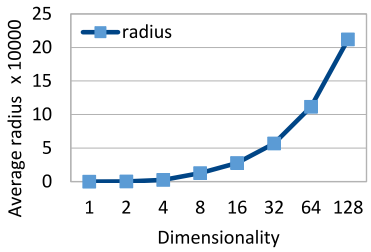
2.1 A Case Study: Brute-Force (BF) Search

We use brute-force search to demonstrate how our data filtering strategy works. To find the k-NN of a given query feature, brute-force search first calculates all the distances between the query feature and all reference features in the database. Then, it uses a max-heap of size k to accumulate the features with the smallest distances. This algorithm is very computationally intensive, with $O(ND)$ cost to calculate the distances and $O(N \log k)$ cost to find the k-NN. Distance calculations will dominate the time, as $\log k$ is very small while D can be large for high-dimensional problems. It also has a large memory footprint, as it needs to scan the whole database for each query.

Algorithm 1. Brute-Force Search with Data Filtering.

```
   Input: X: reference feature database
   Input: q: query feature
   Input: k: number of nearest features required
   Output: heap: max-heap that contains the k-NN
 1 Initialize heap with size k;
 2 heap.max ← ∞;
 3 forall the r_i ∈ X do
 4     tmp_1 ← SED_Estimation(q, r_i);
 5     if tmp_1 < heap.max then
 6         tmp_2 ← SED(q, r_i);
 7         if tmp_2 < heap.max then
 8             heap.push(tmp_2, i);
 9             heap.popMax();
10 return heap with the k nearest features;
```

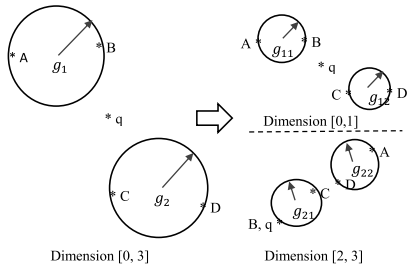
Since k is usually much smaller than N (the size of the database X), many distance calculations are not necessary as most features will be far away from the query feature. If we can exclude those features that are unlikely to be within the k-NN using simple distance estimation, the performance should be improved. As shown in Algorithm 1, instead of calculating the SED directly, the data filtering based search will call the function $SED\_Estimation$ first (line 4), in order



(a) Average radius of a randomly generated dataset with an increasing dimensionality.

	e_1	e_2	e_3	e_4
q	0	0	0	0
A	-4	2	3	4
B	-2	2	1	0
C	2	-3	0	1
D	4	-3	4	3
g_1	-3	2	2	2
g_2	3	-3	2	2

(b) A 4-dimensional case with one query feature and four reference features.



(c) Distance estimation through full-space clustering (left) and multi-subspace clustering (right).

Fig. 1. The challenge of using clustering for distance estimation in high-dimensional spaces. (a) shows the average radius of a randomly generated dataset after clustering. This dataset contains 10,000 features, which are divided into 32 groups using the k-means clustering algorithm. Each element of each of the features is uniformly distributed in the range of [1, 128]. To make it easier to understand, we give a simple four-dimensional example, whose features are listed in (b). (c) shows how our subspace clustering method works with this example.

to quickly estimate the distance between the query feature and reference feature. Only when the estimated distance is smaller than the current maximum distance in the heap will the algorithm calculate their real SED (line 6). There is an implicit assumption that the estimation is always an overestimation. In this way, many unnecessary calculations and memory accesses can be avoided, assuming that the $SED\_Estimation$ calculation is simple, fast and accurate.

2.2 Distance Estimation through Clustering

The key issue now is how to estimate the distances accurately and efficiently. Clustering is a method that has often been used to estimate the distances between one query feature and a group of features. For example, the k-means clustering algorithm in the FLANN library [10] works as follows: first, it randomly chooses C features from X, that will be used as group centers. Then, it calculates the distances between all the other features in X and these group centers. Finally, each feature will be assigned to its closest group, as determined by the distances to group centers. After the first iteration, new group centers can be generated by calculating the mean value of the features within each group. Multiple iterations are needed in order to achieve a good clustering result. In this paper, we use the same k-means clustering algorithm for our data filtering strategy. Although better clustering methods could be used, this would not affect our general approach.

After clustering, the entire feature space is divided into a number of groups and the center of each group is used to represent the features within that group. However, when the dimensionality becomes large, the features will be sparsely distributed in the space and thus the radius of each group will also become large. For example, Fig. 1a gives the average radius of the groups in a randomly generated dataset. As we can see, the average radius of the groups grows quickly with the increasing dimensionality. When the radius is large, traditional clustering-based distance estimation schemes will become less accurate.

Let's take a simple four-dimensional case as an example, as illustrated in Fig. 1b. Here, q is the query feature; and A, B, C and D are four reference features. After clustering on the reference features, based on the all four dimensions, A and B are put into the same group with the center g_1, and C and D are put into the other group with the center g_2. The left side of Fig. 1c illustrates the clustering result (we use circles as a simplified representation of hyperspheres within the actual four-dimensional space). If we use this clustering result to estimate distance between the query and the reference features, then $\|g_1 - q\|$ will represent $\|A - q\|$ and $\|B - q\|$ while $\|g_2 - q\|$ will represent $\|C - q\|$ and $\|D - q\|$. As $\|g_1 - q\| = 21$ and $\|g_2 - q\| = 26$, the order of the reference features based on the distance estimation is A, B, C, D. However, their real distances are $\|A - q\| = 52$, $\|B - q\| = 8$, $\|C - q\| = 15$ and $\|D - q\| = 43$, and the correct order is actually B, C, D, A. If $k = 1$, the k-NN search based on this distance estimation will have 0 percent accuracy, while in the case of $k = 2$, the accuracy is only 50 percent.

From the above example we can find that traditional clustering within high-dimensional spaces has two key problems. First, it is so coarse-grained that it is not able to tell the difference between features within the same group. For example, it cannot tell that B is much closer to q than A. Second, it could easily produce incorrect results: just because a group center is close to a query vector, does not mean that all features in that group are close to the query vector. For example, though group g_1 is closer to q than group g_2, feature C in g_2 has a smaller distance to q than A of group g_1. The reason is that the radius of each group could be very large, and thus this risks obscuring the differences between groups.

To overcome this problem, we apply subspace clustering to the example, which is shown in the right side of Fig. 1c. In the subspace clustering approach, the four-dimensional space is divided into two subspaces based on the first two dimensions and the last two dimensions respectively. Within each of the subspaces, we use the same clustering method to divide the features into two groups. The average radius of the groups in subspace clustering is only 0.75, which is much smaller than the average radius in the full-space clustering, which was 6. Since the radius has reduced in magnitude, the estimation accuracy using clustering can be improved. For high-dimensional spaces, subspace clustering can be even more effective, as a larger number of subspaces can be used. We will demonstrate this in Section 4.

2.3 Subspace Clustering for Filtering

Based on the above analysis, we propose the following Subspace Clustering for Filtering method. As Fig. 2 shows, the data structure of SCF is a multi-level cover of the feature

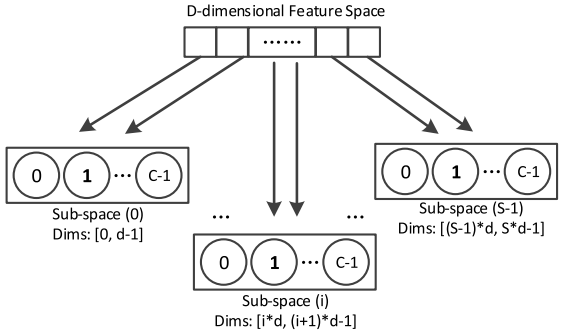


Fig. 2. The basic structure for the SCF method, when it is using S subspaces. All features are assigned to C different groups within each subspace.

space. Instead of using all of the dimensions for clustering, SCF divides the space into S subspaces, each of which has $\lfloor \frac{D}{S} \rfloor$ dimensions. The remainder of $\frac{D}{S}$ can either be treated as an additional subspace, or these dimensions can be distributed to the other subspaces. Then, within each subspace, we use the aforementioned k-means clustering method to divide the features into C different groups where each group will contain $\frac{N}{C}$ features on average. The SCF-based distance estimation depends on two data structures: the SCF index and a matrix of partial distances for the query feature.

The SCF index is created based on the clustering results in the subspaces. The detailed algorithm for creating the SCF index is given in Algorithm 2. β, θ and γ in the algorithm are three matrices that represent the SCF index. Each element β_{ij} $\big(i \in [0, N), j \in [0, S)\big)$ represents the group ID of the ith feature of X within the jth subspace. θ_{jt} $\big(t \in [0, C)\big)$ represents the center of the tth group in the jth subspace. Similarly, γ_{jt} is used to represent the radius of the tth group in the jth subspace.

Algorithm 2. Building the SCF Index.

Input: S: number of subspaces
Input: C: number of groups
Output: $\beta[N][S]$: group IDs of each feature
Output: $\theta[S][C]$: centers of each group
Output: $\gamma[S][C]$: radii of each group
$d \leftarrow \lfloor \frac{D}{S} \rfloor$;
for $i \leftarrow 0$ **to** $S-1$ **do**
  Based on dimensions $[i \times d, (i+1) \times d)$, use a clustering method (e.g., k-means) to divide X into C groups;
  for $j \leftarrow 0$ **to** $N-1$ **do**
    $\beta[j][i] \leftarrow$ group ID of the jth feature;
  for $j \leftarrow 0$ **to** $C-1$ **do**
    $\theta[i][j] \leftarrow$ center of the jth group;
    $\gamma[i][j] \leftarrow$ radius of the jth group;
return β, θ and γ;

The matrix of partial distances for the query feature is created by Algorithm 3. It is represented by the matrix δ in the algorithm. The Partial SED (PSED) between the query feature and group centers in each subspace can be defined as

$$PSED_{l,u}(f_i, f_j) = \sum_{m=l}^{u} \big(f_i[m] - f_j[m]\big)^2, \tag{2}$$

where $1 \le l \le u \le D$, and $[l, u]$ bound the dimensions used to form a subspace.

TABLE 1
PSEDs between q and Group Centers in the Example

	g_{11}	g_{12}	g_{21}	g_{22}
q	13	18	0.5	24.5

Algorithm 3. Calculation of Partial Distances between the Query Feature and the Group Centers.

Input: q: query feature
Input: $\theta[S][C]$: group centers
Output: $\delta[S][C]$: PSEDs between q and θ
$d \leftarrow \lfloor \frac{D}{S} \rfloor$;
$\delta[S][C] \leftarrow 0$;
for $i \leftarrow 0$ **to** $S-1$ **do**
  for $j \leftarrow 0$ **to** $C-1$ **do**
    $l \leftarrow i \times d$;
    $u \leftarrow (i+1) \times d - 1$;
    $\delta[i][j] \leftarrow PSED_{[l,u]}(q, \theta[i][j])$
return δ;

Algorithm 4 shows the steps for distance estimation. The PSED between the query and the center of a group is used to estimate the PSED between the query and reference features within that group. For each reference feature, the sum of all estimated PSEDs in every subspace is used as the Estimated SED (ESED) between the query and the reference feature.

Algorithm 4. $SCF\_Estimation(q, r_t)$

Input: q: query feature
Input: t: the index number of r_t in X
Input: $\delta[S][C]$: PSED matrix
Output: $ESED$: estimated SED
$ESED \leftarrow 0$;
for $i \leftarrow 0$ **to** $S-1$ **do**
  $ESED \leftarrow ESED + \delta[i][\beta[t][i]]$;
return $ESED$

Table 1 shows the matrix for the PSEDs of the previous example, where $g_{11} = (-3, 2, \_, \_)$, $g_{12} = (3, -3, \_, \_)$, $g_{21} = (\_, \_, 0.5, 0.5)$, and $g_{22} = (\_, \_, 3.5, 3.5)$. Thus, in the right side of Fig. 1c, the ESEDs of all reference features are as follows:

$$\begin{aligned}
\|A - q\|_{\text{esed}} &= \|g_{11} - q\|_{\text{psed}} + \|g_{22} - q\|_{\text{psed}} = 37.5 \\
\|B - q\|_{\text{esed}} &= \|g_{11} - q\|_{\text{psed}} + \|g_{21} - q\|_{\text{psed}} = 13.5 \\
\|C - q\|_{\text{esed}} &= \|g_{12} - q\|_{\text{psed}} + \|g_{21} - q\|_{\text{psed}} = 18.5 \\
\|D - q\|_{\text{esed}} &= \|g_{12} - q\|_{\text{psed}} + \|g_{22} - q\|_{\text{psed}} = 42.5.
\end{aligned}$$

They result in the estimated order B, C, A, D, which is closer to the real order of B, C, D, A than that estimated based on the original full-space clustering.

2.4 Space and Time Complexity Analysis

As shown in the above algorithms, SCF uses small index structures. Since there are S subspaces and each one has C

groups, it takes $O(SC\frac{D}{S}) = O(CD)$ space to store all the group centers (θ) and $O(SC)$ space to store the radius of each group (γ). Then, it takes $O(NS)$ space to store group IDs (β) for all reference features. During runtime, it will cost $O(SC)$ space to store the PSEDs (δ) for each query feature. As the magnitude of N will dominate the other parameters, the space complexity for SCF can be simplified as $O(NS)$. Since S is much smaller than D (8 versus 128 in our implementation for the SIFT dataset), the index structure of SCF is more likely to fit into the shared cache. For example, when $N = 20{,}000$, the brute-force algorithm needs to access up to 10 MiB memory (each element of the feature vector is a 32-bit floating point number) while the SCF structure only needs around 160 KiB (the group ID is represented by one byte). Therefore, SCF can better utilize the shared cache and requires significantly fewer memory accesses compared to the original brute-force algorithm.

Compared with the brute-force algorithm, SCF can make time savings by reducing the number of euclidean distance calculations. As Algorithm 4 shows, in SCF, it costs $O(SC\frac{D}{S}) = O(CD)$ time to calculate the PSEDs between the query and all group centers. Then, SCF takes $O(NS)$ time to estimate the distances for all features. Finally, for each query, only a few features will be selected to have their real distances calculated. Suppose c percent of the features will be selected. It takes $c\% \times O(ND)$ to find the k-NN. From our experimental result on SIFT features, SCF only picks $c\% \approx 3.13\%$ of the features. This shows that SCF is very efficient in reducing unnecessary distance computations.

2.5 Discussion

It is worth noting that the overhead of Algorithm 2 is a one-off cost, which will be relatively minor when amortized over many queries. Also note that by adjusting S and C in the above algorithms, we can change the estimation accuracy of SCF. Usually when S and C are increased, the estimation accuracy improves. We will show the correlation between accuracy, S and C, and discuss how to further improve estimation accuracy in Section 4.

3 Evaluation

In this section, we evaluate the performance of our SCF method when it is applied to four popular k-NN algorithms on five real-world and synthetic datasets. The performance improvement that they attain on three multicore platforms is analyzed.

3.1 Experimental Setup

In this section, we detail the hardware and software configurations for our performance evaluation.

3.1.1 Multicore Platforms

Three multicore platforms are used in our evaluation:

1) AMD16: AMD Opteron Processor 8380, 4 cores × 4 sockets @ 2.5 GHz, 6 MiB L3 shared cache, 16 GiB DDR2 (800 MHz) memory;
2) AMD64: AMD Opteron Processor 6276, 16 cores × 4 sockets @ 2.3 GHz, 16 MiB L3 shared cache, 64 GiB DDR3 (1,333 MHz) memory;
3) MIC: Intel Xeon Phi Coprocessor 5110P, 60 cores @ 1.0 GHz, 30 MiB L2 shared cache, 8 GiB GDDR5 (5.5 GHz) memory.

The multicore machines, AMD16 and AMD64, are two typical multicore platforms, which are popular and widely deployed. For example, the Dell R815 (AMD64) is still a popular server being marketed by Dell. MIC has been in mass production since 2013. The g++-4.4 compiler is used on the AMD16 and AMD64 machines while icc-14.0 is used on the MIC platform.

3.1.2 Algorithms

Four popular algorithms for high-dimensional problems are used in our evaluation. We give a short introduction of each of them, here:

1) Brute-force: This algorithm searches the whole database to find the k-NN, as described in previous section. It is effective when dealing with problems that require accurate results.
2) Randomized kd-Trees: This is an efficient variant of the popular kd-tree algorithm. Multiple kd-trees are built as its index structure. During searching, it traverses these kd-trees and puts good candidate nodes in a priority queue for the next round of searching. In this way, it only searches those most promising nodes to save time. It is efficient for acquiring approximate k-NN results.
3) Hierarchical k-means (HKM): This is an efficient variant of the popular k-means clustering algorithm. Instead of clustering the data only once, it recursively divides large top-level groups into smaller sub-groups until they are small enough to form a basic group (e.g., smaller than 32). During searching, it also uses a priority queue to store candidate groups. This algorithm is also very efficient for getting approximate results.
4) Random Ball Cover (RBC): This method uses a two-level clustering to organize its groups. On each level, it uses BF to do efficient searching. There are two variants: one provides exact and one provides approximate results. The approximate version is more efficient for high-dimensional problems and we will use this version in our experimental evaluation.

The implementations of the first three algorithms (BF, RKD and HKM) have been taken from the FLANN [10] library, which is also included in OpenCV [14] to provide fast approximate k-NN search functionality for computer vision related tasks. RBC is the state-of-the-art algorithm on parallel platforms [12] and is well optimized to reduce scalability problems when running on multicore systems.

Two common metrics are used to evaluate their improved performance:

1) Speedup: Sequential execution time divided by parallel execution time: $\frac{T_{sequential}}{T_{parallel}}$;
2) Improvement: Original execution time divided by execution time after applying SCF: $\frac{T_{original}}{T_{SCF}}$.

TABLE 2
Overview of the Test Datasets

Name	Number of ref features	Number of query features	Dimensionality
SIFT	25,271	7,481	128
Random	25,000	7,500	128
Madelon	2,000	1,800	500
HAR	7,352	2,947	560
Digits	3,823	1,797	64

3.1.3 Datasets

The datasets listed in Table 2 are used to evaluate the performance of the aforementioned algorithms:

1) "SIFT" contains features generated by SIFT [7] algorithm, which is one of the most widely used algorithms in computer vision and shows great value in many practical use cases.
2) "Random" is a synthetic dataset, which contains features that are randomly generated. Due to its randomness and uniform distribution, this dataset is challenging for most k-NN algorithms.
3) "Madelon" is an artificial dataset containing data points grouped in 32 clusters placed on the vertices of a five dimensional hypercube [15]. It is one of five datasets used in the NIPS 2003 feature selection challenge.
4) "HAR" is a real-world dataset previously used in [16]. It contains the recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smart phone with embedded inertial sensors.
5) "Digits" is a real-world dataset [17], which contains handwritten digits that have been size-normalized and centered in a fixed-size image (8×8).

3.2 Sequential Performance Improvement

In this section, we evaluate the sequential performance improvement after applying SCF to the aforementioned four algorithms. The results are collected by running them on a single core of the AMD16 machine. Note that part of the performance improvement comes from a sacrifice of precision. Please see Section 4 for further discussion regarding precision; we focus on performance issues in this section.

As shown in Fig. 3, SCF can improve the performance by up to $8.85\times$ for BF (in the "HAR" case) and up to $5.78\times$ for RBC ("SIFT"). This can be explained by the exhaustive search in both algorithms benefiting greatly from SCF. The performance improvements of RKD and HKM are not as good as that for BF and RBC. This is because both RKD and HKM spend a lot of time searching their complex index structures to get a small number of good candidates. Since the number of candidates for filtering is small, SCF has less effect on the two algorithms. However, on average, SCF can still improve the performance of RKD by 33 percent and that of HKM by 19 percent.

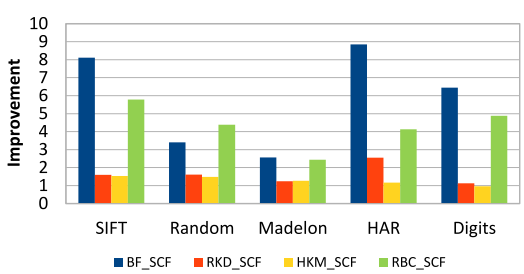


Fig. 3. Sequential performance improvement after applying SCF to each algorithm on the AMD16 machine.

TABLE 3
Parallel Performance Improvement of BF_SCF over the Original BF Algorithm on Each Platform and Dataset when Using All of the Available Cores

Platform	SIFT	Random	Madelon	HAR	Digits
AMD16	$34.75\times$	$12.52\times$	$2.50\times$	$20.91\times$	$3.91\times$
AMD64	$15.54\times$	$5.04\times$	$2.66\times$	$9.43\times$	$4.13\times$
MIC	$3.23\times$	$2.11\times$	$1.43\times$	$2.97\times$	$1.33\times$

3.3 Parallel Performance Improvement

In this section, we evaluate the parallel performance improvement of our method. Here, all algorithms are parallelized by OpenMP and the suffix "_SCF" means that SCF is applied to the corresponding algorithm.

3.3.1 Performance Improvement of the BF_SCF

Table 3 lists the parallel performance improvement of BF_SCF on the AMD16, AMD64 and MIC machines. Compared with their sequential performance shown in Fig. 3, the BF_SCF search has the most improvement. For the case of the SIFT dataset on AMD16, its improvement is $34.75\times$ (16 cores), which is much better than the $8.11\times$ improvement on a single core. Fig. 4a explains why the parallel BF_SCF is able to get more performance gain than its sequential counterpart. The speedup curves in the figure show the good scalability of BF_SCF, while the original BF's speedup curves become flat after eight cores. On both the AMD16 and the AMD64 machines, BF hits the memory wall much earlier than when all cores are used. This result shows that for an embarrassingly parallel algorithm like BF, the memory wall becomes one of the most serious bottlenecks.

However, after applying SCF, its scalability has been significantly improved. For example, the speedup against the original sequential BF has been improved from $3.0\times$ to $104.59\times$ on the AMD16 and from $12.84\times$ to $199.63\times$ on the AMD64 when all cores available are used. On the MIC platform, the scalability of the original BF is better because MIC has much higher memory bandwidth. Moreover, since MIC has four hardware threads in each core, it can efficiently hide the memory latency through overlapping computation and memory access. In this case, the memory wall problem in the original BF is greatly relieved and it has reasonable scalability on MIC, as shown in Fig. 4c. However, BF_SCF still has much better performance than the original BF due to the reduced computation.

The scalability of the proposed SCF method is always better than existing k-NN algorithms, but BF_SCF shows the best scalability. Since the scalability improvement on other algorithms is much smaller (several times) than

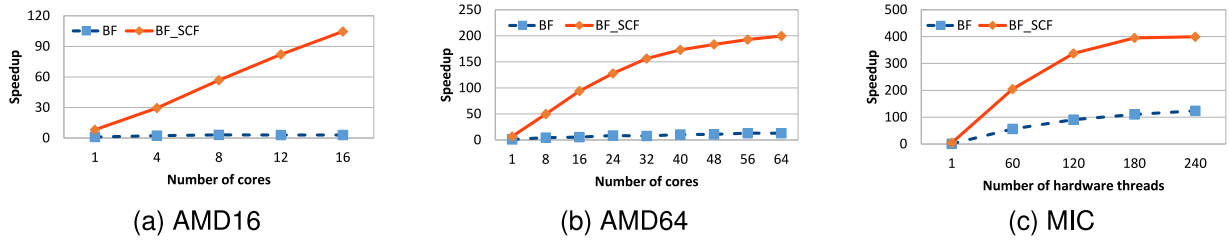


Fig. 4. Scalability of BF and BF_SCF when running on the three platforms. The *y*-axis represents the speedup over sequential BF algorithm. Here the SIFT dataset is used as an example.

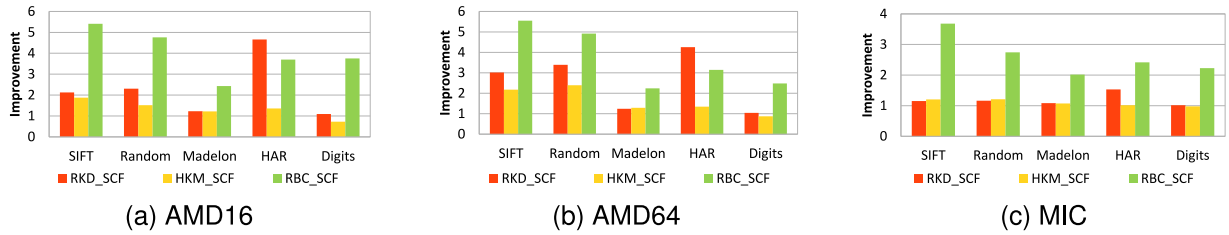


Fig. 5. Parallel performance improvement of other algorithms with different dataset on the three platforms.

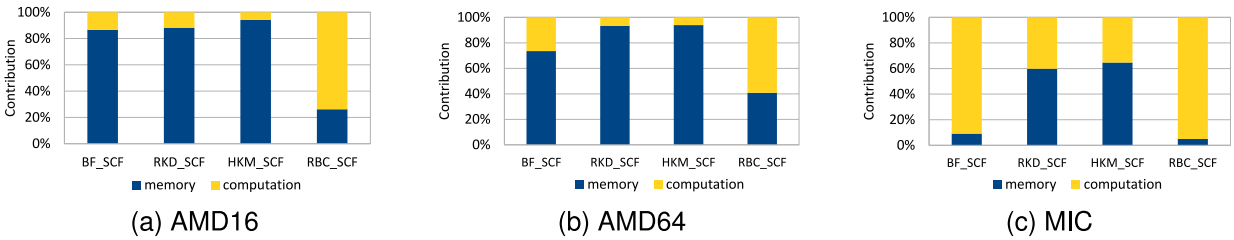


Fig. 6. The percentage of contributions to the performance improvement on the three platforms with "SIFT".

BF_SCF, we would have to use additional figures in order to differentiate their scalability clearly. We have omitted the scalability graphs of SCF applied to other algorithms due to limited space.

3.3.2 Performance Improvement of Other Algorithms

Figs. 5a, 5b and 5c show the performance improvement of other k-NN algorithms on parallel platforms. The performance improvement of RBC_SCF is very similar to that of its sequential counterpart ($5.77\times$ vs. $5.41\times$ on AMD16 and $5.64\times$ vs. $5.54\times$ on AMD64 in the best cases). Since this algorithm has already been optimized for multicore platforms, it scales well on parallel platforms and does not suffer from the memory wall. This shows that SCF is very cache-efficient and has little impact on the performance of those algorithms that already have good cache utilization. On AMD16, RKD_SCF and HKM_SCF get their best performance improvement of $4.66\times$ and $1.87\times$, which is much better than their sequential improvement ($2.55\times$ and $1.53\times$). Similar results are observed on AMD64.

However, for the "Madelon" and "Digits" datasets, neither the RKD_SCF nor HKM_SCF algorithms have more of a performance improvement than their sequential counterparts. The reason is that both datasets are quite small (3.8 MiB for "Madelon" and 0.88 MiB for "Digits") so that they can fit in the last-level cache and are less likely to hit the memory wall. Moreover, due to the lower dimensionality, RKD and HKM perform efficiently on "Digits" anyway. Thus, fewer features can be filtered by SCF. Nonetheless, in most cases SCF can significantly improve performance in these algorithms on AMD16 and AMD64.

Since the MIC platform has a higher memory bandwidth, the memory wall problem is relieved and the performance improvement of most algorithms after applying SCF is quite similar to their sequential counterparts, which means they scale well on this new platform. We note that the current evaluation code does not contain low-level optimizations specific to the architecture, and thus its computing ability may not be fully utilized. We will explore this in our future work.

3.3.3 Contributors to the Performance Improvement

Figs. 6a, 6b and 6c show the contributions of reduced computation and memory accesses to the performance improvement. As mentioned before, these are the two key factors that provide a performance improvement. However, since reduced computation means reduced memory accesses in SCF, precise separation of these two factors is not easy. Therefore, these figures only give a rough idea of their contributions to the overall performance improvement.

On both AMD16 and AMD64, memory is a bottleneck and the reduced memory accesses contribute most of the performance gain of BF_SCF, RKD_SCF and HKM_SCF as they are memory-intensive algorithms. However, as RBC is optimized for multicore platforms by efficiently utilizing the shared cache, reduced memory accesses contribute less in this algorithm.

On MIC it is more obvious that most of the performance gain of RBC_SCF comes from reduced computation. Although BF requires a large number of memory accesses, it can efficiently utilize the high memory bandwidth due to its regular memory access pattern. Therefore, the reduced computation contributes most to the performance gain in

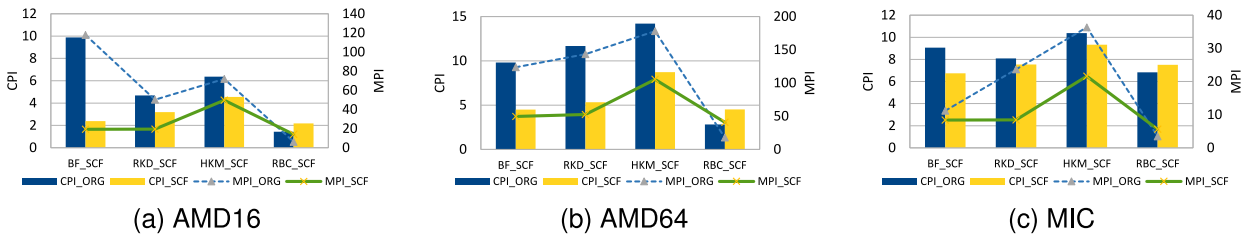


Fig. 7. Statistics of hardware performance monitoring counters on the three platforms with "SIFT".

BF_SCF. However, for RKD_SCF and HKM_SCF, as their index structures have many branches that lead to irregular memory accesses, the reduced memory accesses are still crucial to their performance gain.

3.3.4 Statistics of Performance Monitoring Counters

Figs. 7a, 7b and 7c are provided to verify our previous observations and analyses. In the figures, Cycles Per Instruction (CPI) is used to evaluate the computing efficiency while Misses Per Instruction (MPI) is used to represent intensity of the last-level cache misses per instruction. For both AMD16 and AMD64, the CPIs have a very close relationship with the MPIs as they grow and drop in the same pattern. That means the CPIs are mainly affected by the memory latencies. However, for MIC, CPI is not significantly influenced by MPI, which indicates that the Xeon Phi can provide high memory bandwidth for these algorithms.

4 Performance versus Precision

Quantifying the acceptable level of precision is a relevant issue for approximate k-NN algorithms as there is always a trade-off between search precision and performance. Although 100 percent precision is not necessary in some domains, such as many computer vision use cases, very low precision is definitely not acceptable. In this section, we discuss a few ways for SCF to best balance performance and precision.

4.1 Evaluation of SCF Parameters

The number of subspaces, S, and the number of groups in each subspace, C, are two important parameters within the SCF method. We use two datasets, "SIFT" and "Random", to show the importance of choosing the right (S, C) pair for SCF. Four metrics are used to evaluate the effectiveness of SCF:

1) Estimation Time (ET) represents the total time used for distance estimation (in seconds) by SCF. The smaller ET is, the faster the SCF distance estimation can be;
2) Average Error (AE) is the average absolute error between the real distances (SED) and the estimated distances by SCF. The smaller AE is, the higher SCF's estimation accuracy will be;
3) Filtering Rate (FR) represents the percentage of reference features that can be excluded by SCF. The higher FR is, the more features are excluded;
4) Lost Precision (LP) shows the percentages of k-NN results that are falsely excluded by SCF. The smaller LP is, the better for our filtering strategy.

Since both the two datasets have a dimensionality of 128, we choose S within the range of [1, 64] while C is either 8, 16 or 32. Usually C can be up to 255 if one byte is used to represent the group ID. Although other values for C can be used in SCF, the three values are sufficient for our discussion, given space limitations. Figs. 8 and 9 show the performance and precision of BF_SCF on "SIFT" and "Random" datasets respectively.

From Figs. 8a and 9a, we can find that the ET is highly related with the value of S. This result matches our previous analysis for its time complexity of $O(NS)$. Since distance estimation takes an important proportion of the search time, we should pick S to be as small as possible. However, a small S cannot guarantee a good estimation accuracy. As Figs. 8b and 9b show, when S is small, the AE is very high for both datasets. When S is increased, AE decreases, which shows that the subspace clustering can really improve estimation accuracy. Note that a larger value for C will also reduce AE.

For the "SIFT" dataset, a poor estimation accuracy (high AE) leads to SCF having poor search precision, as we can find in Figs. 8c and 8d. When S is small, FR is kept above 95 percent, which means most distant features can be excluded by SCF. However, the LP can be up to 80 percent, which means many true k-NN features are excluded. With an increased S, FR increases and LP decreases, which shows a reduced AE could lead to better precision. Of course, this sacrifices performance since ET will increase in this case.

For the "Random" dataset, if the estimation accuracy of SCF is poor, this will lead to poor performance, as

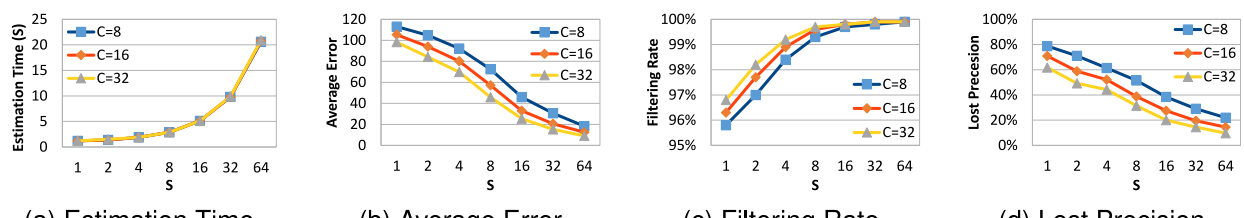
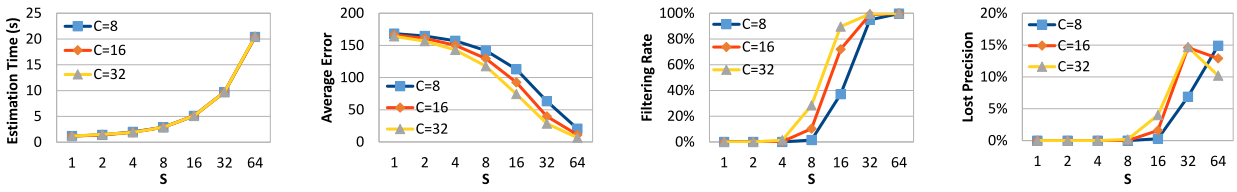


Fig. 8. The performance and precision of the SCF-based brute-force search algorithm using different numbers of sub-spaces (S) and clustering groups (C) on the "SIFT" dataset. The brute-force search time is 40.16 s.



(a) Estimation Time. (b) Average Error. (c) Filtering Rate. (d) Lost Precision.

Fig. 9. The performance and precision of the SCF-based brute-force search algorithm using different numbers of sub-spaces (S) and clustering groups (C) on the "Random" dataset. The brute-force search time is 39.4 s.

demonstrated in Figs. 9c and 9d. Although LP can be as small as 0 percent when S is small, which means no precision is lost, FR is also very low, which means no features are actually excluded. In this case, BF_SCF will have a worse performance than BF since all features will be searched. When S is increased, FR will increase so that SCF can filter more features. However, the precision will be sacrificed as LP will also increase in this case.

Based on the above analysis, we can find that a high value of AE for "SIFT" shows most estimated distances are larger than their real values (worst case: larger than k-NN), which causes false filtering. However for "Random", a high value of AE means most estimated distances are smaller than their real values (worst case: smaller than k-NN), which leads to poor filtering efficiency. Thus, a careful selection of S and C is necessary to keep a good balance between performance and precision for the SCF method.

4.2 Selection of S and C

For BF_SCF, the search time (ST) contains two parts: the total time for distance estimation (ET) and the time for searching the un-excluded features. Suppose the original BF search time is ST_{BF}, we have

$$ST_{BF\_SCF} = ET + (1 - FR) \times ST_{BF}. \quad (3)$$

The performance improvement can be expressed as $\frac{ST_{BF}}{ST_{BF\_SCF}}$. To model the balance between performance and precision, we use a new metric named *efficiency* (ξ) to evaluate SCF, which is the product of performance improvement and precision

$$\xi = \frac{ST_{BF}}{ST_{BF\_SCF}} \times (1 - LP) = \frac{ST_{BF} \times (1 - LP)}{ET + (1 - FR) \times ST_{BF}}. \quad (4)$$

In this case, neither poor performance improvement nor poor precision would lead to good efficiency since good performance with poor precision is unacceptable while poor performance with good precision is meaningless for SCF. We use the data collected in Figs. 8 and 9 to fill (4) and the results are given in Fig. 10. Thus, by picking the parameters when the highest values of ξ are achieved, we have the best (S, C) pair (8, 16) for the "SIFT" dataset and (16, 32) for the "Random" dataset.

4.3 Reward-and-Penalty Strategy

Although selecting the best (S, C) pair is important, the process has some limitations. First, it is time-consuming to search all possible (S, C) pairs. Second, even if the best (S, C) pair is found, the precision may still not be good enough. For example, even if the best (S, C) pair is chosen for "SIFT", its LP is still as high as 38.4 percent.

To solve the above problem so that the FR is kept high and LP is kept low, we propose a Reward-and-Penalty strategy. This strategy is based on the rationale that, if two features are allocated to the same group by the clustering method, they are more likely to be nearest neighbors; otherwise, they are less likely to be nearest neighbors. Therefore, during distance estimation, we will reward those features that are in the close group to the query feature and punish those features in distant groups.

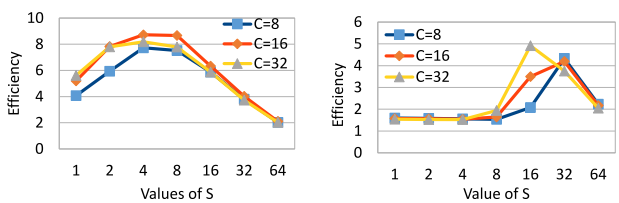
The Reward-and-Penalty strategy is implemented as shown in Algorithm 5. In each subspace, we use $\frac{\delta[i][j]}{\gamma[i][j]}$ (line 3) to tell whether the query is in the corresponding group (line 5). Then, we can update PSEDs by multiplying it by ψ (line 9). The value of ψ is decided by φ_r and φ_p (line 6 or line 8) so that we can control how much we want to reward or punish. It is also related to the value of $(\frac{\delta[i][j]}{\gamma[i][j]} - 1)$ so that closer groups will be rewarded more while more distant groups will be punished more. We call φ_r and φ_p as the reward and penalty factors respectively.

Algorithm 5. Reward-and-Penalty Strategy

Input: $\gamma[S][C]$: radius for each group
Input: $\delta[S][C]$: PSEDS
Input: φ_r: reward factor
Input: φ_p: penalty factor
Output: $\delta[S][C]$: updated PSEDs

```
1  for i ← 0 to S − 1 do
2    for j ← 0 to C − 1 do
3      v ← δ[i][j] / γ[i][j];
4      ψ ← 0;
5      if v ≤ 1 then
6        ψ = 1 + φ_r × (v − 1);
7      else
8        ψ = 1 + φ_p × (v − 1);
9      δ[i][j] ← δ[i][j] × ψ
10 return δ
```

Fig. 11 gives the results after applying this strategy on the two datasets. For the "SIFT" dataset, when φ_r is increased,



(a) SIFT. (b) Random.

Fig. 10. Efficiency of SCF with different S and C.

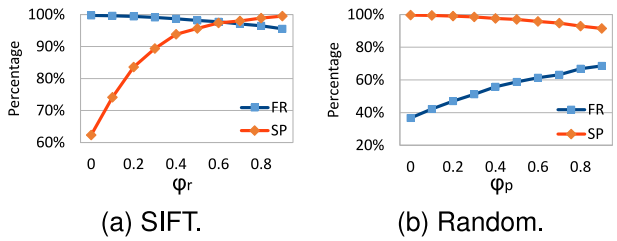


Fig. 11. The influence of Reward-and-Penalty strategy on the FR and SP. Here, $S = 16$ and $C = 8$.

the search precision ($SP = 1 - LP$) also increases from 66 to 98 percent (when $\varphi_r = 0.8$) while FR is still kept above 95 percent. For the "Random" dataset, when the φ_p is increased, its FR increases significantly with little effect on the precision. In this way, the best values for φ_r and φ_p can be decided through this training processing on the sample data.

In summary, the performance and precision of SCF is related to four parameters: the number of subspaces S, the number of clustered groups C, the reward factor φ_r and the penalty factor φ_p. Based on the steps described above, we find their optimal values for each dataset, as listed in Table 4. These values can keep the LP within 5 percent, so that the SP of the target algorithms will not be significantly affected. Additionally, it is worth noting that the above selection process of the optimal values is not part of the k-NN algorithms and they are only used by SCF once decided, so it is not an extra overhead of SCF when used with the k-NN algorithms.

5 Related Work

In this paper, we focus on accelerating high-dimensional k-NN search on multicore platforms. Our proposed SCF method is able to estimate the distance of high-dimensional features with a cache-friendly structure. This paper is an extended version of [18], and contains further details and more experimental results, including introduction of the Reward-and-Penalty strategy, which can greatly improve the precision. As far as we know, this is the first effort on optimization of approximate k-NN algorithms on multicore systems that addresses both performance and precision. Our notion of data filtering based on SCF is inspired by several previous research works.

5.1 Speeding up k-NN on Multicore Systems

Designing multicore-friendly approximate algorithms has been a recent trend for accelerating k-NN search [11], [12]. Good performance is observed with these algorithms on parallel platforms. As more advanced hardware will be deployed in the future, these algorithms have the potential to get a "free ride" for improving performance. Our proposed technique focuses on reducing the accesses to the feature data through filtering. It is general and can be combined with existing algorithms with good precision.

The Xeon Phi is a new coprocessor that uses the Intel Many Integrated Core (MIC) architecture. Currently, many researchers are exploring this new architecture. For example, Heinecke et al. have implemented the famous Linpack Benchmark on it, Liu et al. have designed efficient sparse matrix-vector multiplication on this new architecture [20], and Ramos and Hoefler have studied the communication model for its cache-coherent protocol [21]. As far as we know, our work is the first effort evaluating the performance of k-NN algorithms on Xeon Phi.

TABLE 4
Reward-and-Penalty Strategy Parameter Selection

Name	S	C	φ_r	φ_p
SIFT	8	16	0.7	0.3
Random	16	32	0	0
Madelon	8	16	0	1
HAR	8	16	0.3	0.7
Digits	4	16	1	0

5.2 Latency Optimization and Data Representation

Cong and Makarychev [22] find that hardware threads on several multicore platforms are not sufficient to mask the memory latency of graph algorithms. Perron et al. [23] show that even for an embarrassingly parallel algorithm—seismic imaging—data movement costs can dominate the execution time. Tang et al. [24] show that some popular k-NN algorithms may hit the memory wall due to a poor utilization of the last-level cache. These works highlight potential memory latency issues when dealing with multicore platforms.

Vector Approximation (VA) [25] and Vector Quantization (VQ) [26] share a similar idea of using small structures to represent data. For example, image databases are usually so large that they have to be stored on disk. To reduce the I/O overhead, these algorithms propose a compressed structure to represent the data so that it can be put in memory. In our case, it is more challenging to improve cache utilization as its capacity is much smaller than that of main memory. Reducing time complexity is also very important as memory latency is much smaller than disk latency. While VA uses one dimension and VQ uses full dimensions to build the index, our method can choose any number of dimensions to better balance time complexity and estimation accuracy.

5.3 Dimension Reduction and Distance Estimation

Many datasets that appear to be high dimensional are actually governed by a small number of sub-dimensions. Based on this fact, methods for reducing dimensionality are proposed to discover the dominant sub-dimensions and use them to accelerate the search. Several works and surveys on this topic are available [27], [28], [29], [30]. However, it is often not easy to find the dominant dimensions in high-dimensional spaces and the performance can vary from dataset to dataset. For our SCF approach, we do not need to find these dominant dimensions. Instead, we divide the whole high-dimensional space into several smaller subspaces. Although within each subspace it is not possible to cluster the whole dataset accurately, we can achieve good results with a higher match probability by combining the results from different subspaces.

Location Sensitive Hashing (LSH) is a popular category of algorithms for data mining [31], [32]. By designing special hash functions so that features that are close to each other will get the same hash value, searching can be finished within sub-linear time. However, LSH's precision may not be high enough as it is highly dependent on the hash

function. Also, developing an appropriate hash function for the LSH algorithm can be a very complex undertaking [12].

6 Conclusions

Many k-NN algorithms run into serious bottlenecks caused by the memory wall on multicore systems. In this paper, we propose a data filtering strategy that tries to reduce the number of computation-intensive and memory-intensive distance calculations so that existing algorithms can scale better on multicore platforms. A novel Subspace Clustering for Filtering method is proposed to accurately estimate similarity, i.e., Squared euclidean Distance, between features in high-dimensional spaces. Experimental results show that SCF is general enough to significantly benefit several k-NN algorithms on multicore platforms.

Acknowledgments

The authors thank the anonymous reviewers for their valuable comments. Xiaoxin Tang would like to thank the University of Otago for hosting his PhD internship during the course of this research. This work is partially sponsored by the National Basic Research 973 Program of China (No. 2015CB352403), the National Natural Science Foundation of China (NSFC) (No. 61261160502, No. 61272099), the Program for Changjiang Scholars and Innovative Research Team in University (IRT1158, PCSIRT), the Scientific Innovation Act of STCSM (No. 13511504200), and the EU FP7 CLIMBER project (No. PIRSES-GA-2012-318939).

References

[1] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.

[2] J. Buhler, "Efficient large-scale sequence comparison by locality-sensitive hashing," *Bioinformatics*, vol. 17, no. 5, pp. 419–428, 2001.

[3] D. L. Donoho, "High-dimensional data analysis: The curses and blessings of dimensionality," in *Proc. AMS Math. Challenges Lecture*, 2000, pp. 1–32.

[4] C. Zanchettin, B. L. D. Bezerra, and W. W. Azevedo, "A KNN-SVN hybrid model for cursive handwriting recognition," in *Proc. Int. Joint Conf. Neural Netw.*, 2012, pp. 1–8.

[5] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, "Big data: The next frontier for innovation, competition, and productivity," McKinsey Global Institute, 2011, http://www.mckinsey.com/Insights/MGI/Research/Technology_and_Innovation/Big_data_The_next_frontier_for_innovation

[6] S. Agarwal, Y. Furukawa, N. Snavely, I. Simon, B. Curless, S. M. Seitz, and R. Szeliski, "Building Rome in a day," *Commun. ACM*, vol. 54, no. 10, pp. 105–112, 2011.

[7] D. Lowe, "Object recognition from local scale-invariant features," in *Proc. 7th IEEE Int. Conf. Comput. Vis.*, 1999, vol. 2, pp. 1150–1157.

[8] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proc. 23rd Int.Conf. Mach. Learn.*, 2006, pp. 97–104.

[9] J. M. Kleinberg, "Two algorithms for nearest-neighbor search in high dimensions," in *Proc. 29th Annu. ACM Symp. Theory Comput.*, 1997, pp. 599–608.

[10] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Proc. Int. Conf. Comput. Vis. Theory Appl.*, 2009, pp. 331–340.

[11] M. Al Hasan, H. Yildirim, and A. Chakraborty, "Sonnet: Efficient approximate nearest neighbor using multi-core," in *Proc. IEEE 10th Int. Conf. Data Mining*, 2010, pp. 719–724.

[12] L. Cayton, "Accelerating nearest neighbour search on manycore systems," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2012, pp. 402–413.

[13] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *Proc. 7th Int. Conf. Database Theory*, 1999, pp. 217–235.

[14] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision with the OpenCV Library*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2008.

[15] I. Guyon, A. B. Hur, S. Gunn, and G. Dror, "Result analysis of the NIPS 2003 feature selection challenge," in *Proc. Adv. Neural Inf. Process. Syst.*, 2004, vol. 17, pp. 545–552.

[16] 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 *Proc. Ambient Assisted Living Home Care*, 2012, pp. 216–223.

[17] K. Bache and M. Lichman. (2013). UCI machine learning repository [Online]. Available: http://archive.ics.uci.edu/ml

[18] 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 *Proc. 23rd ACM Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2014, pp. 305–310.

[19] A. Heinecke, K. Vaidyanathan, M. Smelyanskiy, A. Kobotov, R. Dubtsov, G. Henry, A. G. Shet, G. Chrysos, and P. Dubey, "Design and implementation of the Linpack benchmark for single and multi-node systems based on Intel Xeon Phi coprocessor," in *Proc. 27th Int. Symp. Parallel Distrib. Process.*, 2013, pp. 126–137.

[20] X. Liu, M. Smelyanskiy, E. Chow, and P. Dubey, "Efficient sparse matrix-vector multiplication on x86-based many-core processors," in *Proc. 27th Int. ACM Conf. Int. Conf. Supercomput.*, 2013, pp. 273–282.

[21] S. Ramos and T. Hoefler, "Modeling communication in cache-coherent SMP systems: A case-study with Xeon Phi," in *Proc. 22nd Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2013, pp. 97–108.

[22] G. Cong and K. Makarychev, "Optimizing large-scale graph analysis on a multi-threaded, multi-core platform," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2011, pp. 688–697.

[23] M. Perrone, L.-K. Liu, L. Lu, K. Magerlein, K. Changhoan, I. Fedulova, and A. Semenikhin, "Reducing data movement costs: Scalable seismic imaging on Blue Gene," in *Proc. IEEE 26th Int. Parallel Distrib. Process. Symp.*, 2012, pp. 320–329.

[24] X. Tang, S. Mills, D. Eyers, K.-C. Leung, Z. Huang, and M. Guo, "Performance bottlenecks in manycore systems: A case study on large scale feature matching within image collections," in *Proc. 15th IEEE Int. Conf. High Perform. Comput. Commun.*, 2013, pp. 985–995.

[25] R. Weber and K. Böhm, "Trading quality for time with nearest-neighbor search," in *Proc. 7th Int. Conf. Extending Adv. Database Technol.*, 2000, vol. 1777, pp. 21–35.

[26] S. Ramaswamy and K. Rose, "Adaptive cluster distance bounding for high-dimensional indexing," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 6, pp. 815–830, Jun. 2011.

[27] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan, "Automatic subspace clustering of high dimensional data," *Data Mining Knowl. Discovery*, vol. 11, no. 1, pp. 5–33, 2005.

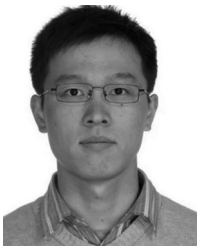
[28] A. Hinneburg and D. A. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," in *Proc. 25th Int. Conf. Very Large Data Bases*, 1999, pp. 506–517.

[29] L. Parsons, E. Haque, and H. Liu, "Subspace clustering for high dimensional data: A review," *SIGKDD Explor. Newslett.*, vol. 6, no. 1, pp. 90–105, Jun. 2004.

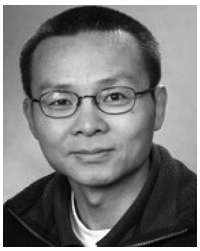
[30] H.-P. Kriegel, P. Kröger, and A. Zimek, "Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering," *ACM Trans. Knowl. Discovery Data*, vol. 3, no. 1, pp. 1:1–1:58, Mar. 2009.

[31] W. Kong, W.-J. Li, and M. Guo, "Manhattan hashing for large-scale image retrieval," in *Proc. 35th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2012, pp. 45–54.

[32] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling LSH for performance tuning," in *Proc. 17th ACM Conf. Inf. Knowl. Manage.*, 2008, pp. 669–678.



Xiaoxin Tang received the BS degree in computer science from the South China University of Technology, China, in 2010. In the year of 2013 and 2014, he was a visiting student in the Department of Computer Science, University of Otago, New Zealand. He is currently working toward the PhD degree at the Embedded and Pervasive Computing Center (EPCC), in the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His research interests include parallel algorithms, heterogeneous computing, high-performance computing, and image matching.



Zhiyi Huang received the BSc and PhD degrees in computer science from the National University of Defense Technology (NUDT), China, in 1986 and 1992, respectively. He is an associate professor at the Department of Computer Science, University of Otago. He was a visiting professor at EPFL and Tsinghua University in 2005, and a visiting scientist at MIT CSAIL in 2009. His research fields include parallel/distributed computing, multicore architectures, operating systems, green computing, cluster/grid/cloud computing, high-performance computing, and computer networks. He has more than 100 publications. He is a member of the IEEE.

David Eyers received the undergraduate computer engineering and maths degrees from UNSW, Sydney, Australia. He received the PhD degree from the University of Cambridge, where he was a senior research associate. After receiving the PhD degree, he joined the University of Otago as a lecturer. His recent research has examined security enforcement and efficient data management mechanisms within wide-area distributed systems. This has included working with event-based middleware, role-based access control, decentralised information flow control, and more recently various cloud technologies. He is a member of the IEEE.

Steven Mills received the BSc and PhD degrees in computer science from the University of Otago, in 2000. After working for a short time in Christchurch as a software developer, he took up a lectureship at the University of Nottingham. In 2006, he returned to New Zealand and worked in commercial research and development at the Geospatial Research Centre and then Areograph Ltd before returning to the University of Otago as a lecturer in 2011. His research interests are in computer vision, and particularly in the reconstruction of 3D scenes from multiple views.

Minyi Guo received the BS and ME degrees in computer science from Nanjing University, China, in 1982 and 1986, respectively, and the PhD degree in information science from the University of Tsukuba, Japan, in 1998. From 1998 to 2000, he was a research associate of NEC Soft, Ltd. Japan. He was a visiting professor at the Department of Computer Science, Georgia Institute of Technology. He was a full professor at the University of Aizu, Japan, and is currently the head of the Department of Computer Science and Engineering at Shanghai Jiao Tong University, China. His primary interests include automatic parallelization and data-parallel languages, bioinformatics, compiler optimization, high-performance computing, and pervasive computing. He is a senior member of the IEEE and has published more than 150 papers in well-known conferences and journals.