J. Vis. Commun. Image R. 44 (2017) 106–115

Contents lists available at ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Massive parallelization of approximate nearest neighbor search on KD-tree for high-dimensional image descriptor matching ☆

Linjia Hu, Saeid Nooshabadi *

Department of Computer Science, Michigan Technological University, Houghton, MI 49931, USA

ARTICLE INFO

Article history:
Received 16 July 2016
Revised 5 January 2017
Accepted 11 January 2017
Available online 12 January 2017

Keywords:
KD-tree
Approximate nearest neighbor search
Parallel algorithm
GPU
CUDA
Image descriptor matching

ABSTRACT

To overcome the high computing cost associated with high-dimensional digital image descriptor matching, this paper presents a massively parallel approximate nearest neighbor search (ANNS) on *K*-dimensional tree (KD-tree) on the modern massively parallel architectures (MPA). The proposed algorithm is of comparable quality to traditional sequential counterpart on central processing unit (CPU). However, it achieves a high speedup factor of 121 when applied to high-dimensional real-world image descriptor datasets. The algorithm is also studied for factors that impact its performance to obtain the optimal runtime configurations for various datasets. The performance of the proposed parallel ANNS algorithm is also verified on typical 3D image matching scenarios. With the classical local image descriptor signature of histograms of orientations (SHOT), the parallel image descriptor matching can achieve speedup of up to 128. Our implementation will potentially benefit realtime image descriptor matching in high dimensions.

1. Introduction

Point descriptors have become popular for obtaining image to image correspondence for 3D reconstruction and object recognition. Search for the image point descriptors that are similar to the query, is one of the core techniques in object recognition and surface registration. To increase the feature descriptiveness, the image descriptors, typically, require high dimensionality [1–8]. However, feature matching in high dimensional space demand extremely high computational workload.

There has been a large body of work in image descriptor matching, exploring the efficient indexing and nearest neighbor search (NNS) in point cloud. A brute force *P*-NNS compares *M* query points with all the *N* points in the search set, to obtain their *P* nearest neighbors. It results in the time complexity of O(*MN*) [9]. Search can be made more efficient by using spatial data structures, such as R-tree, B-tree, quad-tree, binary space partitioning (BSP) tree, K-means tree and *K*-dimensional tree (KD-tree). These structures subdivide the space containing all the points into smaller spatial regions, where a hierarchy is imposed on the smaller regions in a recursive fashion. The NNS on this hierarchical spatial data structure is generally more efficient since it can prune large portions of target dataset.

In 2D/3D point cloud object recognition and perception, NNS require fast performance [10,11]. Unlike the typical applications with single point query [12], the NNS in these point cloud applications involves batch processing a large number of query points to match them against the points in the model object.

The current computing trends favor flexibility of heterogeneous programming model that combines multicore centeral processing unit (CPU) and many-core graphical processing unit (GPU). The GPU, as a typical massively parallel architecture (MPA) complements the CPU through large number of computing cores, and is finding its way into general purpose computing, where fine-grain parallelism is need. Compute unified device architecture (CUDA) and open computing language (OpenCL) standards exemplify this paradigm [13,14]. In particular, GPU has been widely employed for fast and real-time implementation of 3D image processing algorithms [4,15–19]. The inherent massive-parallelism in NNS algorithm can be exploited for implementation on any computing platform that supports fine-grain parallelism.

To mitigate the computational workload associated with high-dimensional digital image descriptor matching, in this paper, we propose a massively parallel approximate *P*-NNS (*P*-ANNS) on GPU to accelerate image descriptor matching. Our parallel *P*-ANNS in all stages is fine-grain parallelized for high-dimensional image descriptor datasets. We employ a hybrid technique which

☆ This paper has been recommended for acceptance by M.T. Sun.

* Corresponding author.
E-mail addresses: linjiah@mtu.edu (L. Hu), saeid@mtu.edu (S. Nooshabadi).

http://dx.doi.org/10.1016/j.jvcir.2017.01.013
1047-3203/

combines non-linear and linear search features. For backtracking we use a priority queue which records the distances to the axis aligned bonding box (AABB). In trading off the efforts in tree traversal and backtracking (due to branch divergence) with the effort in linear search (due to leaf node size or the KD-tree height), our technique finds a near optimal performance point. Moreover, set an upper bound on the number of backtracks for all query points, to reduce the impact of query outliers. Further, all factors that impact the performance are evaluated for the a near optimal configuration of *P*-ANNS. The paper is organized as follows. Section 2 presents the basic concepts of KD-tree construction, NNS on KD-tree, and programming model of CUDA. Section 3 briefly outlines the related works and highlights the innovations in the proposed work. Section 4 presents the design and implementation details of our massively parallel *P*-ANNS on GPU. Section 5 discusses the experimental results. Section 6 concludes the paper.

2. Background

2.1. KD-tree

The KD-tree is a hierarchical spatial partitioning data structure for organizing elements (points) in *K*-dimensional space $\mathbb{R}^K$. KD-tree provides the structure to perform *P*-NNS, with the average and best time complexity of $O(N \log N)$ and $O(N)$, respectively, and a space complexity of $O(N)$ [20]. KD-tree partitions the points in the dataset into axis-aligned cells in a hierarchical fashion, with each cell represented by a node in the tree. Starting at the root of KD-tree, the cells are partitioned into two halves by a cutting hyperplane orthogonal to a chosen partition dimension. Typically, the dimension with the maximum span is selected as the partition dimension, and the split value is chosen as the median. Alternatively, the midpoint between the extreme points in that dimension is chosen as the split mark. Each of the two split cells from the root is then recursively split, in the same manner, into other cells. The recursive branching terminates when the number of points that are contained in a cell is no more than a given upper bound. For a KD-tree with leaf nodes containing only a single point, the height is $\log_2 N$.

2.2. NNS on KD-tree

In NNS problem, given are set *S* of *N* searchable reference points, set *Q* of *M* query points, and a distance metric (*e.g.*, Euclidean, Manhattan and Mahlanobis) in *K* dimensions. In a *P*-NNS, the purpose is to search for the *P* closest points in *S* for each point *q* in *Q*.

For high dimesional feature matching, the most promising approximate indexing structures and NNS algorithms including KD-Tree, K-means tree, and locality sensitive hashing (LSH), are evaluated in [8].

The *P*-NNS on KD-tree can be more efficient since large portions of search region are quickly pruned. Starting from the root node, the search moves down the tree using depth first search (DFS). Once the search reaches a leaf node, '*P* points within this node with the shortest distances to the query point are selected as the initial *P* nearest neighbor candidates. However, the initial candidates may not necessarily be the nearest neighbors to the query point. This requires further search for the best candidates in the neighborhood of this initial cell. In the standard search, implemented through backtracking, a closer subtree is visited prior to visiting the more distant subtree [21]. To avoid visiting the unproductive nodes, in this work we replace the normal queue with a priority queue.

In high-dimensional space, the efficiency of exact search on KD-tree is no better than the brute force technique, as most nodes need to be visited [22,21,23]. Therefore, practical KD-tree based applications perform ANNS [23], by simply setting an upper bound on the number of leaf nodes that can be visited. ANNS can perform order of magnitude faster, with a relatively small number of errors.

2.3. Graphical processing unit (GPU) programming model

In this paper, we use CUDA,[1] to parallelize NNS for high-dimensional image descriptor matching on the KD-tree on the GPU. CUDA employs a single instruction multiple data (SIMD) or a single program multiple threads (SPMT) computing paradigm, where parallel programs are encapsulated in *kernel* functions written in CUDA. All program copies, *viz. threads*, are executed in parallel, independent of each other. Threads are further grouped into a *thread block*. Threads in a thread block have access to a common shared memory. Thread blocks in turn are arranged in a *grid* with a common access to the global dynamic random access memory (DRAM) or cache. The thread blocks are executed on the GPU multiprocessors (with 32 compute cores) in units of 32 threads as a *thread warp*.

In spite of ongoing advances in GPU architecture and programming model, there are severe limitations that make the parallel programming on GPU challenging compared with the multi-core or cluster platforms. Limitations include small runtime stack, cache and shared memory sizes that result in the global memory access latency to be a severe draw on performance. The NNS on the KD-tree cannot be parallelized in a straightforward manner due to its non-linear and divergent nature.

3. Related work and proposed innovation

3.1. Related work

There has been a great deal of work on employing parallel architecture to accelerate NNS in a broad range of areas. These works can be classified into two categories: linear and non-linear searches.

3.1.1. Linear NNS

Linear search algorithms use brute force approach in which the distances between the query point in *Q* and the reference points in *S* are computed in parallel. Then, a sequence of parallel scan on GPU is deployed to locate the point with the shortest distance. The parallel implementations of these linear search algorithms on GPU are straightforward. The expected time complexity of these parallel algorithms is $O(N^2/C)$, where *C* is the number of available cores that can execute in parallel. The work in [25] applied a parallel linear search method for photon mapping to locate the nearest photons in the grid and compute the radiance estimation at any surface location in the scene. In [26] points were stored as textures on GPU memory, and three program fragments were used to compute Manhattan distances, and then perform reductions to find the minimum distance. The work in [27] implemented a bucket sort on GPU to partition 3D points into cells. A parallel linear search was used to find the best matches for query points in the buckets.

The work in [28] used an octree and proposed to deploy shifted sorting to sort both query and reference points on the GPU. It used Morton codes to order the octree cells [29,30]. The NNS was implemented through multiple iterations of shifted sorting on GPU. These works focus on the domain of the computer graphic applications with three-dimensional datasets. The data structures employed were adapted to specific needs of the applications.

The work in [12] used an R-tree and proposed a traversal algorithm for multidimensional range query that converts recursive

[1] Compute Unified Device Architecture (CUDA) [24] is a `C/C++` extension parallel programming model created by NVIDIA for GPU platforms.