Efficient Large-scale Approximate Nearest Neighbor Search on the GPU

Patrick Wieschollek[1,4], Oliver Wang[2], Alexander Sorkine-Hornung[3], Hendrik P.A. Lensch[1]

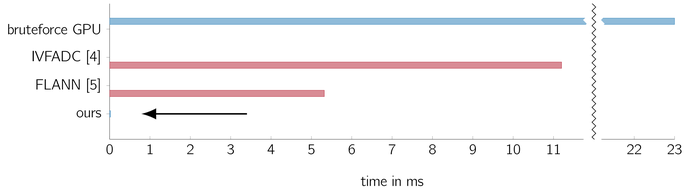
[1] University of Tübingen, [2] Adobe Systems Inc., [3] Disney Research, [4] Max Planck Institute for Intelligent Systems, Tübingen

Overview

Given a large amount of high dimensional data, e.g. 1 billion SIFT features (128GB). Approximate nearest neighbor (ANN) search aims to find

$$N(y) = \arg\min_{x \in \mathcal{X}} \|y - x\|_2^2$$

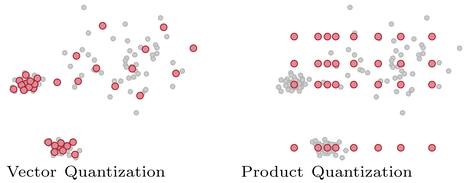
for a query y. Using the GPU, a new re-ranking method and a tree-based index-structur for reducing the exact vector comparisons by factor 122, the total query time is decreased drastically. The proposed method (PQT) allows solving high-dimensional, large scale ANN problems in time critical real-world applications, like loop-closing in videos on a GPU.

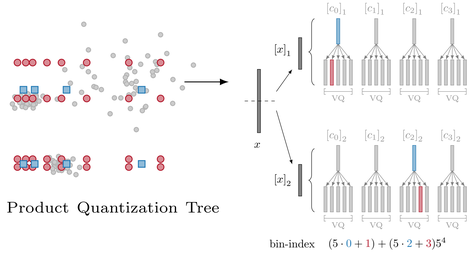


Evaluated on 1 million vector benchmark set for CPU ■ and GPU ■ .

Index-Structure

A common approach is to partition the dataset, e.g. using K-means to find K centroids ● (vector quantization). This has also been done on vector parts, where bucket-combinations across parts produces bins in the product space (product quantization). We propose a tree-based index structure (level 1 ■ , level 2 ●) over all bins. In practise, a codebook of 32 vectors allows us to create **4 trillions** bins instead of 1 million bins as in previous approaches. After pruning, finding the best bin only requires 80 full vector comparisons.



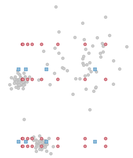


Algorithm

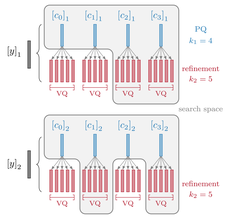
The algorithm consists of an offline and online phase, where the offline phase is pre-computed only once.

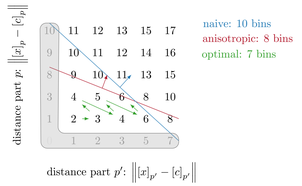
Offline phase Using K-means we cluster a subset of the database to generate the index structure as follows:

```
procedure offline
    ▷ generate index-structure
    sample subset D' ⊆ D from data set
    ▷ generate index-structure
    cluster each part of D' independently into buckets ■
    for each part-bucket of ■ do
        refine part into buckets ● (VectorQuantization)
    ▷ fill index-structure
    for each vector x in data set D do
        j ← calculate id from index-structure
        put v into bin with id hash(j)
        for each part [x]_p of x do
            extract reranking information (λ, c_i, c_j)
```



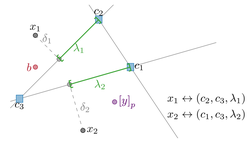
Online phase During the query, only a fraction of second-level clusters are considered. This reduces the actual number of vector comparisons on the SIFT-1M dataset compared to previous approaches from 24576 (which would take 0.13 ms on the GPU) to 200 comparisons for real-world data sets. Our entire query takes only 0.02 ms!





Find candidate bins: To identify good bins in the full space, we need to merge the per-part information of sorted part-buckets. A pre-computed heuristic merges pairs of parts (see above, right), which produces similar results to the optimal sequential solution using Dijkstra's algorithm.

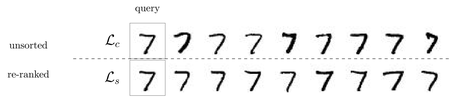
Re-ranking candidate vectors: We collect at most n vectors from the sorted bin sequence in $\mathcal{L}_C$. After sorting based on approximate distances, we drop vectors erroneously placed into the bins from hashing.
Encoding the approximate position of x, i.e., projecting each part $[x]_p$ onto the nearest line through two first-level clusters, only requires (c_i, c_j, λ_k) which is already computed in the offline phase.



The approximation error in $\|y - x\|_2^2$ is bounded by the projection errors δ_i. All vectors in $\mathcal{L}_C$ are sorted by their approximated distance

$$\|y - x\|_2^2 = \sum_{p=1}^{P} \|[y]_p - [x]_p\|_2^2 = \sum_{p=1}^{P} \|[y]_p - f(c_i, c_j, \lambda_k)\|_2^2.$$

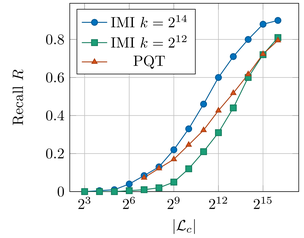
This is the first parallel re-ranking approach which is tailored to GPU architectures.



Parallel re-ranking applied to raw MNIST data.

Results

Accuracy Given a list of candidate vectors, the search quality is measured with recall $R@x$, ie. the proportion of query vectors for which the nearest neighbor is ranked in the first x positions.
Compared to Inverted-Multi-Index [1] (IMI) we achieve nearly the same recall on the 1-billion SIFT dataset with *much* faster query times (0.15ms/0.025 with/without reranking compared to 49ms). Without any re-ranking, the recall of the unsorted list $\mathcal{L}_C$ is illustrated in the following figure.



Timing The time for creating the index structures in previous CPU-based methods has to be measured in days whereas the entire offline-phase of our pipeline only takes 80min on the GPU for 1 billion SIFT features.

method	ms	R@1	R@10	R@100	speedup
FLANN [5]	5.32	0.97	-	-	× 9.6
LOPQ [3]	51.1	0.51	0.93	0.97	× 1
IVFADC* [4]	11.2	0.28	0.70	0.93	× 4.5
PQT (CPU)	4.89	0.45	0.86	0.98	× 10.4
PQT (GPU)	0.02	0.51	0.83	0.86	× 2555
GPU brutef.	23.7	1	1	1	× 2

Total querytime on the standard benchmark set SIFT1M.

References

[1] A. Babenko and V. S. Lempitsky. *The inverted multi-index.*, CVPR 2012.
[2] T. Ge, K. He, Q. Ke, and J. Sun. *Optimized product quantization for approximate nearest neighbor search.*, CVPR 2013.
[3] Y. Kalantidis and Y. Avrithis. *Locally optimized product quantization for approximate nearest neighbor search*, CVPR 2014.
[4] H. Jegou, M. Douze, and C. Schmid. *Product quantization for nearest neighbor search*, TPAMI, 2013.
[5] M. Muja and D. G. Lowe. *Scalable Nearest Neighbor Algorithms for High Dimensional Data*, TPAMI, 2014.

goo.gl/4Z15xB