Efficient Large-scale Approximate Nearest Neighbor Search on the GPU

Patrick Wieschollek1,4, Oliver Wang2, Alexander Sorkine-Hornung3, Hendrik P.A. Lensch1
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 (128x256). Approximate nearest neighbor (ANN) search aims to find

For a query $y$, using the GPU, a tree-based re-ranking and a brute-force structure 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 three critical real-world applications, like large-scale video on a GPU.

Algorithm

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

Offline phase

Using K-means on cluster a subset of the database to generate the index structure as follows:
- procedure offline
  1. generate index-structure
  2. sample subset $D'$ of $D$ from data set
  3. generate index-structure
  4. cluster each part of $D'$ independently into buckets
  5. for each part $v$ of $D'$
     - refer part into buckets $c (VectorQuantization)$
     - extract re-ranking information $(\lambda, c, c, \lambda)$
  6. return index-structure

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 with. Our entire query takes only 0.02 ms.

Find candidate list. To identify good lists in the full space, we need to merge the per-part information of sorted part-buckets. A pre-computed heuristic mergers 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 $k$ vectors from the sorted bin sequence in $C_{\lambda}$.

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

Results

Accuracy

Given a list of candidate vectors, the search quality is measured with recall $R_{i,j}$, i.e., the proportion of query vectors for which the nearest neighbor is ranked in the first $i$ positions.

Compared to inverted Multi-Index (IMI) we achieve nearly the same recall on the 1 billion SIFT dataset with much faster query times (0.13ms/0.02s with without re-ranking compared to 49ms).

Without any re-ranking, the recall of the unsorted list $C_{\lambda}$ 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 80ms on the GPU for 1 billion SIFT features.

References


Index-Structure

A common approach is to partition the dataset, e.g. using K-means to find centroids. We propose a tree-based index structure (level 1, level 2 $\times 2$ use of bins. In practise, a codebook of 32 vectors allows on create 4 trillions bins instead of 1 million bins as in previous approaches. After pruning, finding the best list only requires 80 full vector comparisons.

Vector Quantisation

Product Quantisation

Product Quantisation Tree

Table 1: Query time on the standard benchmark set SIFT1M.

<table>
<thead>
<tr>
<th>Method</th>
<th>R@1</th>
<th>R@10</th>
<th>R@100</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU brutef.</td>
<td>23.7</td>
<td>1.0</td>
<td>0.70</td>
<td>1.00</td>
</tr>
<tr>
<td>IVFADC*</td>
<td>11.2</td>
<td>0.28</td>
<td>0.70</td>
<td>0.93</td>
</tr>
<tr>
<td>PQT (GPU)</td>
<td>4.60</td>
<td>0.45</td>
<td>0.86</td>
<td>3.60</td>
</tr>
<tr>
<td>PQT (GPU) (CPU)</td>
<td>0.02</td>
<td>0.51</td>
<td>0.86</td>
<td>205.00</td>
</tr>
</tbody>
</table>