HM-ANN: Efficient Billion-Point Nearest Neighbor Search on Heterogeneous Memory

Jie Ren¹  Minjia Zhang²  Dong Li¹

¹PASA Lab, University of California, Merced  ²Microsoft

The full paper is accepted in the 34th conference on Neural Information Processing Systems (Neurips'20)
Billion-point nearest neighbor search is challenging.

Billions of data points

Indexing (offline)

Search (online)

\( \mathbb{R}^d \) \( d \sim 100 \)

\( k \)-nearest neighbors

Document retrieval

Image retrieval
State-of-the-art approximate nearest neighbor search algorithms

Graph based methods (HNSW)

- Indexing builds a navigable graph over base points
- Search starts from a specific point and finds the nearest neighbors based on connections in the navigable graph

*Figure courtesy of the “Image Retrieval in the Wild” tutorial in CVPR’20
State-of-the-art approximate nearest neighbor search algorithms

Graph based methods (HNSW)

- Indexing builds a navigable graph over base points
- Search starts from a specific point and finds the nearest neighbors based on connections in the navigable graph

*Figure courtesy of the “Image Retrieval in the Wild” tutorial in CVPR’20*
State-of-the-art approximate nearest neighbor search algorithms

Graph based methods (HNSW)

- Indexing builds a navigable graph over base points
- Search starts from a specific point and finds the nearest neighbors based on connections in the navigable graph

☑ High recall and low latency

☒ Large memory consumption

Construct the graph hierarchically

This structure works pretty well for real-world data

[Figure courtesy of the “Image Retrieval in the Wild” tutorial in CVPR’20]
## State-of-the-art approximate nearest neighbor search algorithms

<table>
<thead>
<tr>
<th>Graph based methods (HNSW)</th>
<th>Compression based methods (FAISS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Indexing builds a navigable graph over base points</td>
<td>• Compressed representations of the database points are used to fit into memory; cluster compressed data</td>
</tr>
<tr>
<td>• Search starts from a specific point and finds the nearest neighbors based on connections in the navigable graph</td>
<td>• Search the closest clusters to the query</td>
</tr>
</tbody>
</table>

- ✔ High recall and low latency
- ✔ Memory friendly
- ✗ Large memory consumption
- ✗ Low recall and high latency
State-of-the-art approximate nearest neighbor search algorithms

Graph based methods (HNSW)
- Indexing builds a navigable graph over base points
- Search starts from a specific point and finds the nearest neighbors based on connections in the navigable graph

Compression based methods (FAISS)
- Compressed representations of the database points are used to fit into memory; cluster compressed data
- Search the closest clusters to the query

ANN with SSD (DiskANN)
- Maintains a copy of compressed data in memory
- re-rank search results using full-precision coordinates stored on SSD

✓ High recall and low latency
✓ Memory friendly
✓ Mem friendly and high recall

✗ Large memory consumption
✗ Low recall and high latency
✗ High latency
State-of-the-art approximate nearest neighbor search algorithms

Graph based methods (NSG, HNSW)
- Indexing builds a navigable graph over base points
- Search starts from a specific point and finds the nearest

Compression based methods (FAISS)
- Compressed representations of the database points are used to fit memory; cluster compressed data

ANN with SSD (DiskANN)
- Maintains a copy of compressed data in memory
- re-rank search result using full-precision coordinates

Can we have high recall, low latency, and low memory cost?

- High recall and low latency
- Large memory consumption

- Memory friendly
- Low recall, high latency

- Mem friendly and high recall
- High latency
Heterogeneous memory is promising

<table>
<thead>
<tr>
<th>Memory Type</th>
<th>Latency (ms/μs/ns)</th>
<th>Bandwidth (MB/s/GB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDD</td>
<td>7.1</td>
<td>2.6</td>
</tr>
<tr>
<td>SSD</td>
<td>68</td>
<td>250</td>
</tr>
<tr>
<td>PMM</td>
<td>170-300</td>
<td>39</td>
</tr>
<tr>
<td>DRAM</td>
<td>100</td>
<td>64</td>
</tr>
</tbody>
</table>

- Intel Optane DC persistent memory module (PMM) is ~80 faster than SSD
  - ANNS with SSD is not designed to enjoy the faster memory access of PMM.

- PMM is 3x slower than DRAM
  - The existing ANNS with a naive data placement strategy can hurt search efficiency badly without considering performance difference between DRAM and PMM.
Combine memory heterogeneity with data heterogeneity

How to reduce the number of memory accesses in the bottom layer (in PMM)?

build high-quality upper layers
make most memory accesses happen in fast memory
HM-ANN

- HM-ANN takes both memory and data heterogeneity into consideration, and enables billion-scale similarity search on a single CPU-based node without using compression.

  - For two billion-sized datasets, HM-ANN achieves 95% recall in less than 1 ms on CPU.

- Search: HM-ANN achieves logarithmic search time complexity.

- Indexing: HM-ANN passes over the dataset twice with a cost of O(Nlog(N)).
## HM-ANN: Indexing

Indexing: output a navigable graph $G$ with high search quality

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td><strong>Top-Down Insertions</strong> (the <strong>first pass</strong> of the dataset): build graph incrementally to build a hierarchical graph $G$ with $L$ layers. Nodes in the bottom layer ($L_0$) represent all points in the dataset</td>
</tr>
<tr>
<td>2.</td>
<td><strong>Sorting</strong> all nodes in $L_0$ based on their degrees</td>
</tr>
<tr>
<td>3.</td>
<td><strong>Deleting</strong> upper layers in $G$</td>
</tr>
<tr>
<td>4.</td>
<td><strong>Bottom-up Promotion</strong> (the <strong>second pass</strong> of the dataset):</td>
</tr>
<tr>
<td></td>
<td>1. Promote the highest-degree nodes from $L_0$ to $L_1$, and the number of nodes in $L_1$ is decided based on the fast memory size (DRAM size).</td>
</tr>
<tr>
<td></td>
<td>2. Promote the highest-degree nodes from $L_i$ to $L_{i+1}$ , with a constant promotion rate</td>
</tr>
</tbody>
</table>
HM-ANN: Indexing - Top-Down Insertions
HM-ANN: Indexing - Top-Down Insertions
HM-ANN: Indexing - Bottom-up Promotion

Fast memory

Slow memory
HM-ANN: Indexing - Bottom-up Promotion
HM-ANN: Search

Search: Minimize searches in slow memory

1. **1-greedy search** from the top layer to $L_2$ to find an entry point $e$ in $L_1$

2. **Search in L1** returns $efSearch_{l_1}$ candidates as entry points for $L_0$

3. **Parallel search in** $L_0$ with multiple threads

4. Return k nearest neighbors
HM-ANN: Search in $L_1$
HM-ANN: Search in $L_1$
HM-ANN: Search in $L_0$
HM-ANN: Search in $L_0$

$L_0$

entry point 3

Nearest Neighbor

entry point 2

query

entry point 1
Evaluation Setup

• Testing bed.
  – Intel Xeon Gold 6252 CPU@2.3GHz.
  – DDR4 (96GB) as fast memory and Optane DC PMM (1.5TB)

• Evaluation metrics
  – The query response time
  – The accuracy with top-K recall (e.g., K=1, or 100), which measures the fraction of the top-K retrieved by the ANNS that are exact nearest neighbors

• Baselines
  – two state-of-the-art billion-scale compression-based methods (IMI+OPQ and L&C)
  – the state-of-the-art graph-based methods (HNSW and NSG).
  – Baselines are measured with Memory Mode
HM-ANN: fast and highly accurate billion-scale ANNS

**top-100 recall of > 90% within 4 ms**

Query time vs. recall curve for top 100 recall

**top-1 recall of > 95% within 1 ms**

Query time vs. recall curve for top 1 recall
HM-ANN: fast and highly accurate billion-scale ANNS

HM-ANN outperforms state-of-the-art compression-based methods in recall-vs-latency by a large margin, obtaining 46% higher recall under the same search latency.
Compare HM-ANN with system-level data management solutions

Explicitly managing data for HM as HM-ANN is the key to achieve superior latency and recall results.

HM-ANN outperforms HNSW with Memory Mode and first-touch NUMA by 2x and 3.7x
The effectiveness of HM-ANN indexing

Without HM-ANN indexing, the elements of L1 in HNSW are selected randomly and sparse, and the entry point found through L1 search are sub-optimal.

HNSW with parallel L0 search only slightly outperforms HNSW.
HM-ANN Performance Breakdown

- Bottom-up promotion (BP)
- Parallel L0 search (PL0)
- Data Prefetching (DP)

- Bottom-up promotion slightly impacts search efficiency.
- Bottom-up promotion, together with the parallel L0 search, significantly improves the search efficiency, compared with running HNSW on HM without explicit data management.

Performance benefit of memory management techniques in HM-ANN
Conclusions

• Releasing full power of heterogeneous memory sometimes requires a co-design of algorithm and system.

• HM-ANN maps the hierarchical design of the graph-based ANNs to memory heterogeneity in Optane-based heterogeneous memory.

• Combined with a set of system-level techniques, HM-ANN avoids expensive accesses in slow memory without sacrificing accuracy.
  – On two billion-scale datasets, HM-ANN provides 95% top-1 recall in less than one millisecond