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

Jie Ren¹, Minjia Zhang² and Dong Li¹
¹University of California Merced, ²Microsoft

Motivation
Enable fast and highly accurate billion-scale ANNS.

<table>
<thead>
<tr>
<th>State-of-the-art ANNS</th>
<th>Graph-based ANNS</th>
<th>Quantization-based ANNS</th>
<th>SSD-based ANNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>HMSW [NG]</td>
<td>IM+OPQ [L&amp;C]</td>
<td>DiskANN [N]</td>
</tr>
<tr>
<td>High Recall</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Low Latency</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Memory support for billion-point</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Heterogeneous Memory (HM) is Promising
- HM = fast memory + slow memory
- Fast mem (e.g., the traditional DRAM)
  - expensive > fast > small mem. capacity
- Slow mem (e.g., Optane PMM)
  - cheap > slow > large mem. capacity

- Challenge 1: The slow memory (PMM) performs ~80X times faster than SSD, but it is still 3X slower than DRAM. The existing ANNS with a naive data placement strategy can not fully enjoy the benefit of HM.

- Hierarchical Graph-based ANNS (HNSW)
  - Each layer is a navigable small world graph.
  - $L_0$ contains all database elements, and the upper layers are randomly selected, nested subsets of database element.
  - The majority of search happens in $L_0$.

- Challenge 2: Can we take both memory and data heterogeneity into consideration, and enables billion-scale ANNS without using compression?

Overview
HM-ANN is a HM-aware graph-based ANNS, which achieves 95% top-1 recall in less than 1ms on a single CPU node.

Algorithm
- **Indexing**: Build navigable graphs with high search quality using two passes over the dataset.
  - **Top-down insertion**:
    - Create navigable small world graph as the bottom-most layer in slow memory.
  - **Bottom-up promotion**:
    - Prioritize promoting pivot points with high degree from the bottom layer graph to form upper layers placed in fast memory.

- **Search**: Make the majority of search happens in fast memory and minimize searches in slow memory.
  - **Upper layer search**
    - Goal: achieve high quality search in fast memory.
    - From the top layer to $L_1$: one-greedy search
    - In $L_1$: $L$-greedy search; prefetch data in $L_0$ into fast memory.
  - **Parallel search in $L_0$**
    - Goal: minimize searches in slow memory while achieving high recall.
    - In $L_0$: multiple entry point with one-greedy searches in parallel.

Experiment Results
HM-ANN establishes the new state-of-the-art for indexing and searching billion-point datasets.

- **Billion-scale algorithm comparison**:
  - Testing bed
    - Intel Xeon Gold 6252 CPU@2.3GHz.
    - DDR4 (96GB) as fast memory and Optane DC PMM (1.5TB)
  - Baselines
    - We extend state-of-the-art graph-based to billion-scale ANNS (HNSW [1] and NSG[2]) on HM through using fast memory as the cache of slow memory.
    - We build two state-of-the-art billion-scale quantization-based methods (IM+OPQ[3] and L&C[4]) with HM.

- **Experiment Results**
  - HM-ANN achieves top-1 recall of > 95% within 1 ms
  - HM-ANN is 2X faster than HM-unaware graph-based methods to reach the same accuracy.
  - HM-ANN outperforms state-of-the-art quantization-based methods in recall-vs-latency by a large margin, obtaining 46% higher recall under the same search latency
  - HM-ANN achieves high search efficiency by performing Bottom-up promotion (BP), Parallel search in $L_0$ (PL0) and Data Prefetching (DP).

References

Techniques in HM-ANN
- Fig. Query time vs. recall curve for top-1 recall for (a) DeepLR, and (b) Baseline
- Techniques in HM-ANN
  - HM-ANN establishes the new state-of-the-art for indexing and searching billion-point datasets.