CptS 580 Computer Vision Bonus work

Sharath Kumar Karnati 011852253

6th December 2024

Questions

Question 1: (25 pts): What are the primary challenges of implementing exhaustive search and exact indexing in billion-scale databases, and why are they impractical?

Answer

Exhaustive search and exact indexing in billion-scale databases face several challenges, making them impractical:

1. **High Computational Cost:** Exhaustive search involves compareing every query to all data base vectors. For large datasets, this process is computationaly expensive, especially when the data has many dimensions, as it requires billions of comparisons.
2. **Memory Limitations:** Storing billions of high dimensional vectors in memory requires significant space. This often exceds the capacity of standard hardware, especially when using full precision, making it hard to process data efficiently.
3. **Slow Query Response Times:** Because exhaustive search scans all entries in the database, the time taken to respond to a query increases linearly with the database size. This delay is unacceptable for real time applications like recomendations or image retrieval.
4. **Scalability Problems:** Handling massive datasets requires advanced hardware, including GPUs or distributed systems. This adds complexity and cost, making it hard to scale exact methods to billions of data points.
5. **Curse of Dimensionality:** In high dimensional spaces, the differences between similar and dissimilar points become less clear. This reduces the effectivness of distance calculations, making searches less meaningful.
6. **Energy and Cost Inefficiency:** The resources needed for exhastive searches at scale result in high energy consumption and operational cost, which are not sustainable for large AI systems.

In summary, the computational demands, memory needs, and slow performance of exhaustive search and exact indexing make them unsuitable for billion-scale datasets. As a result, methods like FAISS focus on approximate nearest neighbor (ANN) searches to provide faster and more efficient solutions.

Question 2: Why is it challenging to perform k-nearest neighbors (k-NN) search efficiently on GPUs?

Answer

Performing k-nearest neighbors (k-NN) search efficiently on GPUs is challenging due to several factors:

1. **High Computational Requirements:** k-NN searches involve computing distances between the query vector and all database vectors. For high dimensional datasets, this results in billions of operations, which can quickly over whelm GPU resources.
2. **Memory Bandwidth Limitations:** GPUs rely on high speed memory, but accessing large datasets stored in off-chip memory can create a bottle neck. Transferring billions of vectors to and from GPU memory slows down the computation.
3. **Efficient Parallelism:** While GPUs excel at parallel processing, k-NN search tasks do not always map well to GPU architecture. For example, load balancing becomes an issue when some threads finish computations earlier than others, leading to underutilized GPU cores.
4. **Sorting Challenges:** Identifying the k-nearest neighbors requires sorting distances for every query. Sorting large arrays is computationally expensive, and GPUs struggle to handle these operations efficiently due to limited shared memory.
5. **Scalability for Large Datasets:** Large datasets cannot always fit in GPU memory, requiring data to be partitoned or transferred in chunks. This increases latency and complicates the implementation.
6. **Precision and Numerical Stability:** GPUs often use floating-point arithmetic optimized for speed rather than precision. For highdimensional and large scale searches, this can lead to slight inaccuracies in distance computations, which may affect the search results.

In summary, the main challenges for efficient k-NN search on GPUs include computational intensity, memory bandwidth limits, difficulties in paralelization, and sorting overhead. These issues require specialized algorithms and optimizations, like those implemented in FAISS, to leverage GPUs effectively.

Question 3: (25 pts): What data structures does FAISS use to improve the efficiency of k-selection on billion-scale databases?

Answer

FAISS uses advanced data structures to improve the efficiency of k-selection on billion scale databases. The key data structures are:

1. **Inverted File Index (IVF):** FAISS partitions the dataset into multiple clusters using a coarse quantizer. Each query is mapped to a subset of clusters, significantly reducing the number of database points that need to be searched for k selection.

2. **Product Quantization (PQ):** To reduce memory usage and computation, FAISS compresses high dimensional vectors into lower dimensional representations. These compact codes enable efficient approximate distance computations, accelerating the search process.

3. **HNSW (Hierarchical Navigable Small World) Graphs:** For fast approximate nearest neighbor search, FAISS employs HNSW graphs. These graphs organize data points into a navigable network that allows quick traversal to the k nearest neighbors, avoiding the need for exhaustive comparisons.

4. **GPU-Specific Structures:** FAISS utilizes GPU optimised data structures such as tiling and blockbased memory access patterns. These structures maximize parallelism and memory bandwidth usage, ensuring efficient processing of large scale data.

5. **Flat Indexes (Optional):** For smaller datasets or exact searches, FAISS uses flat indexes where every vector is stored without compression. While not ideal for billion-scale datasets, this structure is used in conjunction with other methods for hybrid solutions.

In summary, FAISS leverages a combination of inverted indexes, product quantization, graphbased search structures, and GPU optimizations to perform efficient k-selection on billion-scale databases.

Question 4: How does FAISS achieve a balance between efficiency and search accuracy, and what options does it offer for improving the efficiency of k-selection?

Answer

FAISS achieves a balance between efficiency and search accuracy by combining advanced approximation techniques with scalable data structures. It offers several options to improve the efficiency of k selection:

1. **Product Quantization (PQ):** FAISS compresses database vectors into compact codes, enabling approximate distance calculations. While this reduces memory usage and speeds up the search, it introduces a small tradeof in accuracy.

2. **Inverted File Index (IVF):** FAISS clusters the dataset into partitions using a coarse quantiser. Queries are limited to a subset of relevant clusters, drastically reducing the search space and improving efficiency. Users can control the number of clusters to balance speed and accuracy.

3. **Hierarchical Navigable Small World (HNSW) Graphs:** HNSW graphs are used for approximate nearest neighbor searches, where data points are connected in a graph structure. FAISS provides options to tune graph parameters (e.g., the number of connections per node) to adjust the tradeof between accuracy and speed.

4. **Search Parameters:** FAISS allows users to fine tune search parameters such as the number of probes used during cluster search. Increasing the number of probes improves accuracy at the cost of additional computation.

5. **GPU Optimizations:** FAISS leverages GPU acceleration with data structures designed for parallel processing. This increases efficiency without significantly impacting accuracy.

6. **Hybrid Indexes:** FAISS supports combining exact search methods (e.g., flat indexes) with approximate methods like IVF and PQ. This hybrid approach lets users prioritize accuracy for specific use cases while maintaining efficiency.

In summary, FAISS achieves efficiency by reducing the search space, compressing data, and optimizing algorithms for hardware acceleration. Users can customize parameters like cluster count, graph connectivity, and probing depth to find the optimal balance between search speed and accuracy.