Accelerating Milvus on Arm: SVE-Enhanced Vector Search for RAG Applications

Varun Reddy Thatikonda[1], Mohak Goel[1], Adarsh Srivastava[1], Abhishek Jain[1], Nukariya Atushi[1], Doteguchi Masahiro[2], Priyanka Sharma[1]

[1] Fujitsu Research of India Pvt. Ltd. (FRIPL), [2] Fujitsu Limited

Abstract

- Efficient **Retrieval-Augmented Generation** (RAG) relies heavily on fast, scalable vector search. While **Milvus**[1] is a leading solution, its performance on Arm CPUs is often limited by suboptimal utilization of advanced SIMD (Single Instruction, Multiple Data) vector instructions.
- Our work presents a robust solution: Arm Scalable Vector Extension - **SVE**[3] optimized distance kernels integrated directly into **Knowhere**[2], Milvus' core vector search engine.
- This innovation delivers up to **2x faster** retrieval for critical index types, directly translating to significantly higher query throughput, lower retrieval latency, and greater hardware efficiency for RAG workloads. All optimizations are fully upstreamed to the open-source codebase and are readily usable on SVE-capable Arm CPUs.

Methodology

Milvus[1] vector search, fundamental to RAG, relies on index traversal, candidate selection, and **distance computation**. **The latter often become the bottleneck for query performance on Arm CPUs, primarily due to underutilized SIMD capabilities** for common index types, e.g., **IVF_FLAT, IVF_SQ8, HNSW, and AutoIndex.**

To address this critical limitation for RAG, we have implemented **Arm SVE-optimized L2 and Inner Product distance kernels directly within Knowhere**[2], Milvus' core vector search engine. This targeted integration, **leveraging SVE's advanced vector processing, naturally aligns with the compute-intensive nature of distance evaluation in RAG.**

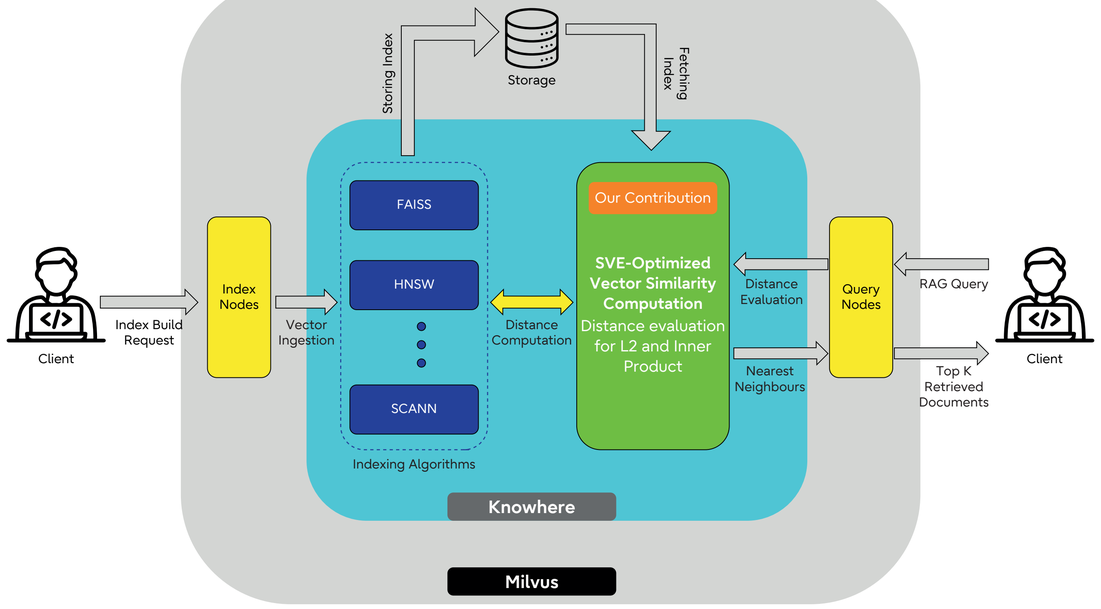

***Fig 1** : Milvus accelerates RAG retrieval by executing SVE-optimized vector similarity kernels within Knowhere's query and indexing paths [1,2,3].*

SVE-Optimized L2 Distance Computation

Inputs:
x, y (FP32 vectors), d (dimension)
Algorithm:
1. Initialize vector accumulator acc ← 0
2. Determine SVE vector length VL
3. For i = 0 to d in steps of VL:
   - (a) Generate predicate mask pg for valid lanes (i < d)
   - (b) Load vector elements using predicate
     x_vec ← ldr(x + i, pg)
     y_vec ← ldr(y + i, pg)
   - (c) Compute vector difference
     diff ← x_vec − y_vec
   - (d) Accumulate squared distance using fused multiply-add
     acc ← acc + diff × diff
4. Reduce vector accumulator across lanes
5. Return scalar L2 squared distance

End Algorithm

***Algorithm 1:** Pseudocode for SVE-optimized[3] FP32 L2 distance function. A similar approach has been used to extend SVE support to Inner Product(IP) metric functions and to FP16, BF16, INT8 data types.*

This **bottom-up approach elevates data-level parallelism, minimizes per-query compute overhead**, and ensures efficient utilization of Arm vector hardware, all while preserving correctness and portability.

Results

All benchmarks were performed using **VectorDBBench**[4], the official benchmarking tool for Milvus, on an **AWS Graviton3** instance with **1M vectors**. We evaluated performance on **Cohere Dataset (768D)** and **BioASQ Dataset (1024D)** to capture the performance across varying dimensionalities. For each dataset, we measured **Queries Per Second(QPS)** to evaluate retrieval performance and **Indexing Time** to evaluate indexing efficiency.

We compared the default Arm execution path against the **SVE-optimized**[3] **implementation**, highlighting performance gains in both retrieval and indexing workloads relevant to RAG applications.

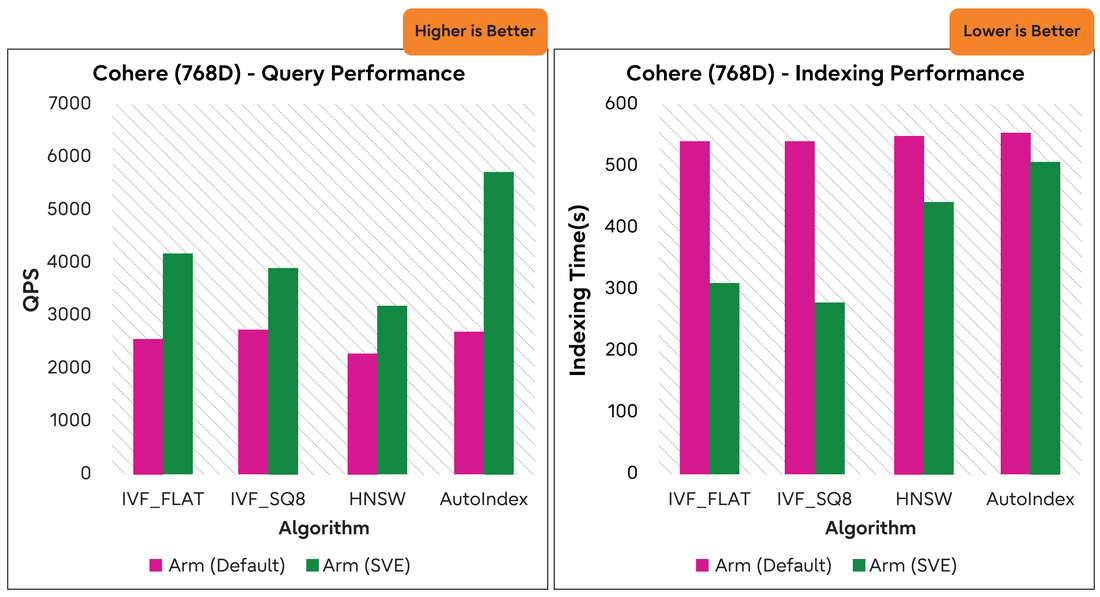

***Fig 2:** Performance comparison on the Cohere (768D) dataset: (a) query throughput and (b) indexing time between default Arm and SVE-optimized implementations.*

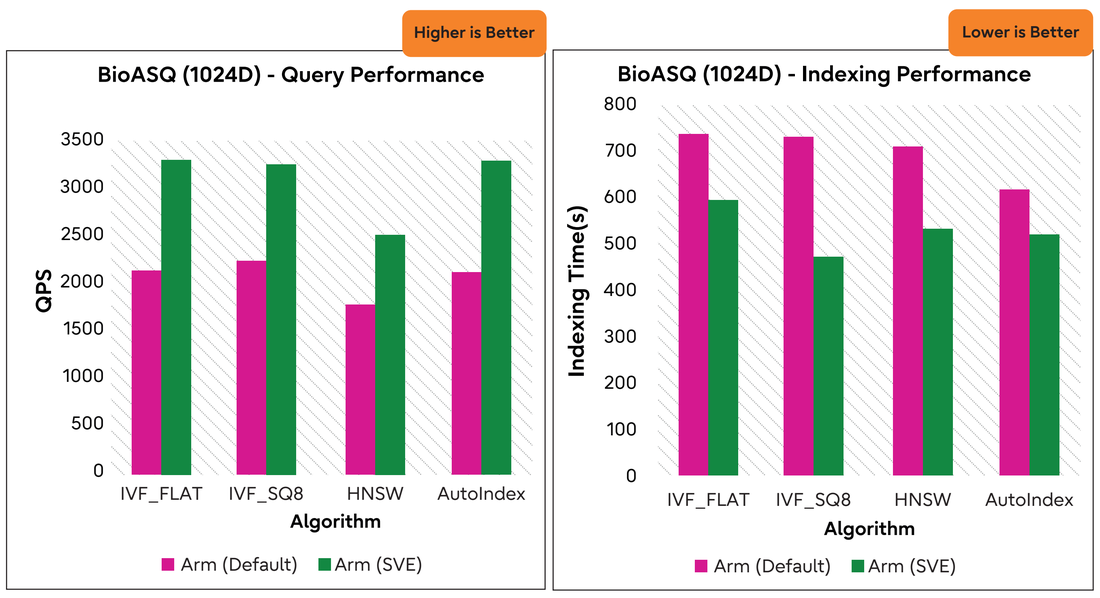

***Fig 3:** Performance comparison on the BioASQ (1024D) dataset: (a) query throughput and (b) indexing time between default Arm and SVE-optimized implementations.*

Conclusion and Future Work

- We integrated Arm SVE–optimized[3] distance kernels into Milvus[1], achieving up to **~2x speedup** (as shown in Figs. 2 and 3) in indexing and query workloads. These optimizations accelerate similarity computation in retrieval-heavy AI and RAG applications.
- While Milvus exposes FP32, FP16, BF16, and binary vectors at the API level, some indexing paths still rely on FP32 internally. We will extend end-to-end SVE support for reduced-precision vectors to eliminate conversions and improve performance on Arm CPUs.

References

[1] Milvus: A Vector Database for Similarity Search, [https://milvus.io/]

[2] Knowhere: Vector Execution Engine, [https://github.com/milvus-io/knowhere]

[3] Arm SVE Instruction Set Architecture, [https://developer.arm.com/documentation/ddi0596/2021-06/SVE-Instructions]

[4] VectorDBBench: An Open Source Vector Database Benchmark Tool, [https://github.com/zilliztech/VectorDBBench]

Links to Pull request

L2-FP32 IP-FP32 L2-FP16 IP-FP16 L2-INT8 IP-INT8 L2-BF16 IP-BF16

This poster is based on results obtained from a project, JPNP21029 subsidized by the New energy and Industrial Technology Development Organization (NEDO).