用于近似最近邻向量搜索专门架构的复现

摘要

向量的相似性搜索或称为最近邻搜索是在向量数据库中检索与所提供的查询向量最相似的一组向量的任务。长期以来，它一直是许多应用程序的关键内核。近年来，随着现代神经网络和机器学习模型将图像、视频和文档的语义表示为称为嵌入的高维向量，向量的近邻搜索变得尤其重要。为所提供的查询嵌入向量找到一组相似的嵌入向量现在是现代推荐系统和语义搜索引擎的关键操作。由于从数十亿个向量中准确地搜索最相似的向量是一项艰巨的任务，因此在许多现实世界的用例中经常使用近似最近邻搜索（ANNS）。不幸的是，研究发现使用服务器级 CPU 和 GPU 来执行 ANNS 任务的性能和能耗效率并不理想。为了解决这些问题，研究人员提出了一种名为 ANNA（Approximate Nearest Neighbor search Accelerator）的专门架构，它与最先进的 ANNS 算法（例如 Google ScaNN 和 Facebook Faiss）兼容。通过结合专用数据流流水线和高效的数据重用的优势，ANNA 比传统 CPU 或 GPU 实现了数个数量级的能耗效率提高和吞吐量提高。

关键词：相似性搜索；硬件加速器；近似最近邻搜索；乘积量化

1 引言

机器学习技术的最新进展使计算机能够在语义上理解查询，从而能够有效地检索最相关的结果。各种现代机器学习和神经网络算法将实体表示为高维嵌入向量。例如，Word2Vec 将单词表示为密集向量，语义相似的向量在向量空间中位置更近。同样，可以将句子、图像、视频等各类数据使用向量表示。通过这种嵌入向量，检索与查询相似的实体的任务就变成了在向量空间中查找一组接近查询嵌入向量的向量。此任务通常称为相似性搜索或最近邻搜索。NNS 最流行的应用是推荐系统，例如，YouTube 推荐系统首先利用 NNS 来识别特定用户的一组候选视频，然后使用单独的重型深度神经网络来选择最佳推荐。类似地，NNS 通常用于有效识别点击率预测模型的一组候选者，例如 Facebook DLRM 、Google DCN 和阿里巴巴 BST。此外，相似性搜索用于传统搜索引擎，例如 Microsoft Bing，也可用于多媒体搜索服务，通过用户提供的图像或音频输入来查找相似的搜索引擎。相似性搜索的广泛适用性促使学术界和工业界的许多研究人员探索各种相似性搜索算法。例如，Facebook 设计并维护了一个名为 Faiss 的相似性搜索库，Google 也发布了一个名为 ScaNN 的相似性搜索库。微软、雅虎等也有自己的类似搜索库。然而，准确地检查数据库中的每个向量并计算查询向量和候选向量之间的相似度需要大量的计算和内存访问。因此，大多数现有的相似性搜索实现都利用近似最近邻搜索（ANNS）算法，该算法需要更少的计算和内存访问，但代价是搜索精度略有下降。

最近的一些工作探索了优化和完善基于压缩的相似性搜索的方法，并且 Google 和 Facebook 等主要行业参与者已经将其基于压缩的相似性搜索库作为开源软件发布。考虑到不断增加的数据规模，研究人员预计基于压缩的方案在可预见的未来将继续成为最主要的方案。

然而，研究人员发现，在传统硬件（CPU 或 GPU）上运行 ANNS 方案并不理想，原因主要是其特定的计算模式需要大量进行存储访问。对于 CPU 来说，效率低下的原因有两个：(1) 应用程序缺乏对片上高速缓存使用的控制，导致难以在高速缓存中保留频繁重用的数据，从而导致了更大量的内存访问，(2) CPU 支持动态控制流或动态提取指令级并行性的能力会给 ANNS 方案带来额外的能量开销，而 ANNS 方案具有相对静态的控制流和易于提取的并行性。另一方面，在 GPU 上，(1) 共享内存使用限制了 GPU 并行性并导致资源利用率不足，(2) 低算术强度的计算模式导致 GPU 计算能力的利用严重不足。

为了解决上述问题，研究人员提出了一种专门用于高效处理基于压缩的相似性搜索算法的专用硬件 ANNA [9]。ANNA 经过精心设计，支持各种基于压缩的相似性搜索算法，因此与现有的软件库（如 Facebook Faiss 和 Google ScaNN）直接兼容。ANNA 与使用 CPU 和 GPU 的 ANNS 实现相比，在十亿级向量数据库中查询相似向量的性能大大提高。

2 相关工作

目前研究人员已经提出了多种 ANNS 算法，主要包括基于哈希的算法、基于图的算法和基于压缩的算法。在这些解决方案中，基于压缩的解决方案是十亿级搜索场景中最受欢迎的选择。基于图的搜索和基于哈希的搜索对于百万级搜索（例如，查找类似的电影）非常有效。但对于更大规模的数据库，它们的内存需求通常会超出服务器通常的主存大小。而基于压缩的方案通常可以在内存中容纳必要的数据，因为它们只需要保存数据集的压缩版本和轻量级索引结构。

近几年也出现了一些针对 ANNS 设计的专用硬件加速器，包括针对 KD 树、LSH 方法以及 PQ 方法的专门电路。但它们大都并非适用于数十亿规模的数据集。

2.1 ANNS 的算法优化方法

软件 ANN 是一个已经经过深入研究的主题，有广泛的相关工作。基于图的 ANNS 算法 [6, 11] 利用合理构造的图结构快速完成查询，在百万级数据集上实现了高性能，但它们对于十亿级搜索来说是不适用的，因为它们需要将占用大内存空间的图驻留在内存中。还存在其他 ANNS 技术，例如利用树结构 [12] 或局部敏感哈希 (LSH) 的技术 [3, 13]，但这些算法的性能目前的表现并不具有优势。

基于乘积量化的 ANNS 算法存在多种变体，旨在提高码本质量。提高码本质量对于提高搜索性能也至关重要，因为高质量的码本需要检查较少数量的数据库向量，同时保持较高的查全率。为此，ScaNN 利用新颖的目标函数 [5]，DPQ 利用基于深度学习的训练 [8]，OPQ 对原始数据库应用旋转 [4]。ANNA 可以在支持这些方法，因为它们没有改变基于乘积量化的搜索和计算模式。ANNA 的实现可能需要稍微扩展，以支持未来某些基于 PQ 的相似性搜索算法，但我们预计 ANNA 设计的核心概念和组件仍然不需要大量改动，因为几乎所有基于乘积量化的相似性搜索算法都以类似的方式操作：加载编码数据，利用查找表简化与编码数据的相似性计算，并选择返回前 k 个候选项。

2.2 基于专用硬件的 ANNS 方法

Abdelhadi 等人 [1] 利用 FPGA 上的大型片上存储器，为基于 PQ 的 ANNS 设计专门的 FPGA 实现。该设计在百万级数据集上实现了高吞吐量，其压缩向量适合片上缓冲区，但并不容易适用于十亿级数据集。Zhang 等人 [16] 还提出了 FPGA 加速的 ANNS，它在 SIFT1M 数据集上实现了 50K QPS，召回率为 0.94 (1@10)。然而，对于大多数十亿级数据集，用于此结果的配置无法实现超过 90% 的高召回率。两种 FPGA 实现都缺乏数据重用优化，因此无法有效利用有限的片外存储器带宽。此外，他们使用为他们的硬件定制的自己的 ANNS 机制。这些自定义机制的指标性能并未像 ScaNN 和 Faiss 等广为人知的软件实现那样经过严格验证。一些先前的工作研究了在硬件上加速 NNS, 但它们与 ANNA 针对的任务不同。例如，Tigris [15] 是一部关于加速 KD 树 ANNS 的学术著作。所提出的实现特定于点云配准任务，其中涉及 3 维数据的 NNS。已知基于 KD 树的 ANNS 对高维数据的搜索精度非常有限，因此所提出的硬件不能用于对每个向量具有超过 100 维的十亿级嵌入进行相似性搜索。此外，Sun 等人 [14] 针对类似的任务提出了一种 DSVS 算法和基于 FPGA 的 3 维 NNS 设计。与 KD 树 ANNS 一样，所提出的 DSVS 算法不适合高维数据。Danopoulos 等人 [2] 利用基于高级综合的硬件来加速 kmeans 聚类算法中的通用矩阵乘法（gemm），该算法用于需要在搜索过程之前执行的 NNS 索引构建。具体来说，它的目标是加速获得质心向量的过程，而 ANNA 的目标是获得质心向量和码本完成后发生的搜索过程。因此，这项工作以及其他 gemm 加速器与 ANNA 所针对的任务不同，不适用于基于 PQ 的 ANNS 查询处理。

3 本文方法

3.1 本文方法概述

本文提出了一种近似最近邻搜索的专用硬件架构。用于加速基于压缩的 ANNS 算法实现。本文还设计了针对内存访问瓶颈的访存优化技术，通过提高数据的重用次数降低内存访问的开销。

3.2 专用于向量的近似最近邻搜索的硬件架构

专用于向量的近似最近邻搜索的硬件架构主要包括三部分,分别为聚类/码表处理器 (CPM)、数据库向量预取器 (EFM)、相似度计算器 (SCM)，基本对应了聚类选择、查找表构造和相似度计算三个部分。其中聚类/码表处理器用于使用朴素的方法进行向量相似度计算, 用于进行聚类选择和使用码表进行查找表构造. 数据库向量预取器用于预取数据库中的向量, 从而避免从内存中读取向量造成的阻塞。相似度计算器用于使用查找表计算查询向量与数据库向量的相似度，并进行 top-k 选出最近邻向量。上述专用架构如图 1 所示：

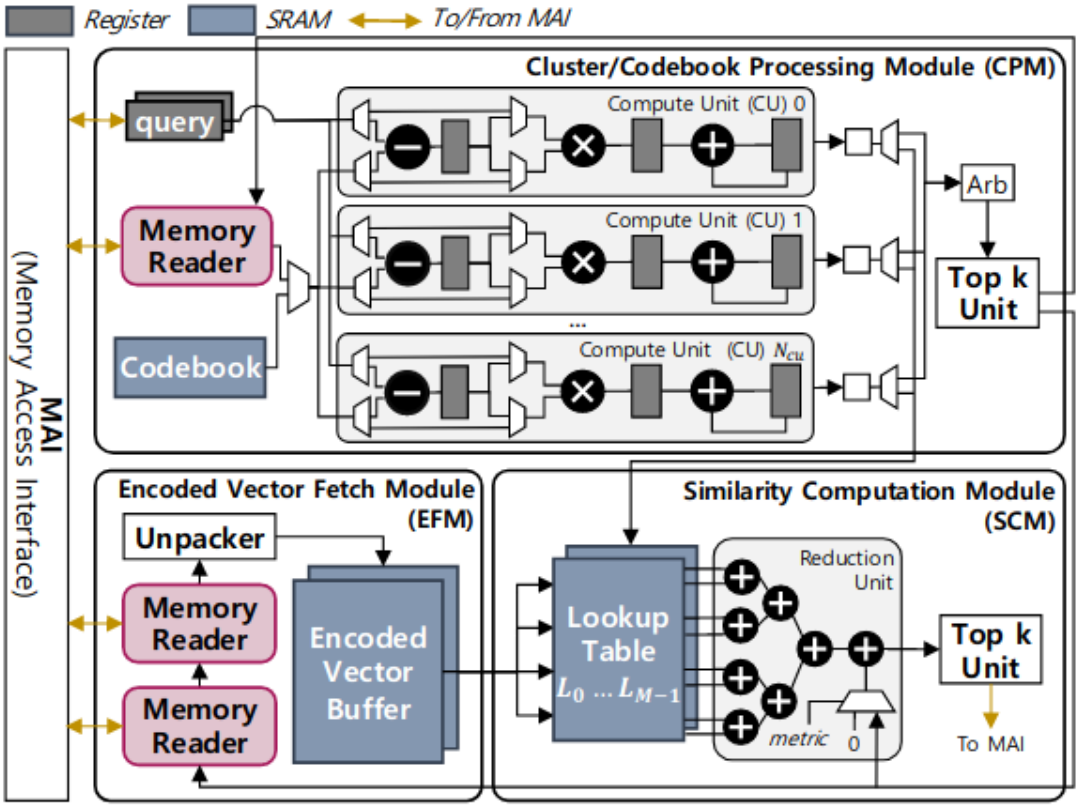

图 1. ANNA 架构示意

具体来看，在专用硬件上进行近似最近邻搜索的流程如下。首先，查询向量与内存中的聚类中心向量在 CPM 中进行逐个元素的朴素距离计算，并通过 top-k 单元获得其中 k 个距离最近的聚类的编号。聚类的编号被传递到 EFM 中进行向量数据库的读取，并缓存到缓冲区中。同时，查询向量与码表在 CPM 中进行查找表的计算，对于 L2 距离，此时还需要聚类中心再次参与计算。最后，数据库中预取出的向量经过查找表和加法树进行距离计算，并经过 top-k 组件选出其中相似度最高的 k 个向量写入内存。

为了实现流水线，向量数据库的缓冲区和 SCM 中的查找表都使用了双缓冲区的技术，使得一个缓冲区被读取端使用时写入端就可以开始写入下一组数据。

3.3 访存优化技术

可以发现，上述架构中，比较容易通过增加各模块数量的方式提高其计算性能，从而匹配其在高负载场景下的计算开销。但与此同时，增加硬件资源却难以显著增加从内存中读取数据的效率，从而使得内存带宽成为制约该专用硬件架构的性能瓶颈。为了有效减轻内存瓶颈对整体架构的制约，本文设计了基于批处理机制的访存优化技术，进一步优化对向量数据库的访存效率。其主要思想是尽可能使从内存中读取的一组向量可以被多个查询所使用，从而减少内存访问的需求。

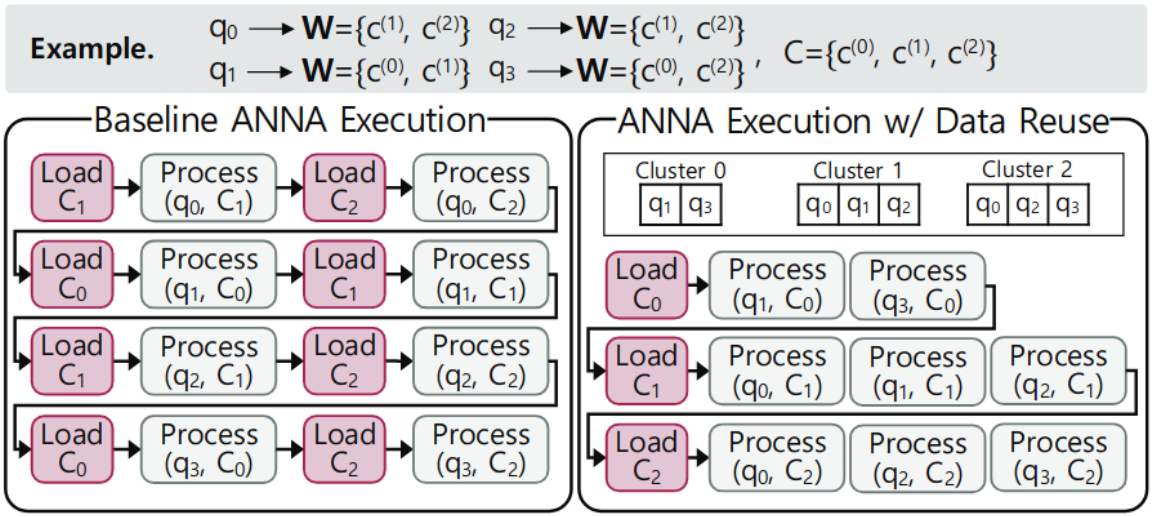

图 2. 基于批处理的访存优化示意

如图 2 所示，C 为每个请求最终需要从内存访问的数据库向量的聚类. 绿色箭头左侧为未进行优化时的流程，此时每个聚类在读取后只被一个查询向量使用，随后就被丢弃. 绿色箭头右侧为优化后的情形，此时先等待若干个请求到达，然后在一次批处理中进行响应. 如图中例子所示，此时每个聚类都有若干个查询需要使用，因此对每个聚类，从内存中进行读取后依次完成每个查询所需要的处理流程，然后丢弃该聚类进行下一个聚类的处理. 由于图中的 Process 阶段容易通过增加硬件资源的方式并行化，而 Load 阶段受限于内存的固有带宽限制难以大量并行化，因此真实情况下访存优化的效果可能比图 2 中更加显著.

在经过访存优化后，一个需要额外处理的情况是：由于每个请求可能在时间上分为几个部分被处理，因此单一的 top-k 组件无法保留使用它的全部查询的 top-k 信息，在不同的聚类处理过程中同一个查询也可能被分配到不同的 top-k 组件上. 一种处理方式是每次将当前的 top-k 结果写回内存中，并在每次使用前再从内存中读取. 由于大多数应用场景需要的 k 的值都不太大，这个新增加的内存访问的开销是可以接受的.

经过并行化，并使用了上述 top-k 写回方法后的处理流程示意图如图 3 所示，可以看出处理流程比较充分的发挥了硬件并行化和流水线化的优势，比较有效地提高了处理效率.

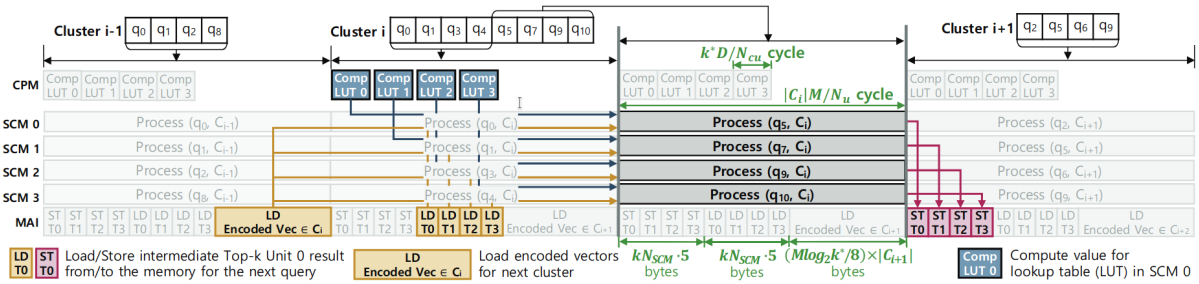

图 3. 经过访存优化的处理流程示意

4 复现细节

4.1 与已有开源代码对比

复现过程中使用了开源的 dramsim3 [10] 作为内存部分的模拟器，在此基础上完成了 ANNA 专用硬件部分的周期级模拟器。ANNA 架构的三个主要模块均为独立完成，没有参考任何相关源代码。在此基础上，我还实现了原文中并未详细介绍的多个 ANNA 的分布式架构，并独立实现了内存交叉访问模块用于分布式架构中异地的内存访问。

4.2 实验环境搭建

在 CPU 和 GPU 平台上，我们使用 FAISS [7] 作为基线，评估在 CPU 和 GPU 平台上 ANNS 算法的性能。Faiss 是一个专门为高效的大规模嵌入检索任务而设计的软件库，较高效地实现了各类 ANNS 算法。FaissCPU 运行在配备 256GB DDR4 内存和双路 Intel Xeon Silver 4210R CPU 的主机系统上，FaissGPU 运行在双路 NVIDIA A100 GPU 上，每个 GPU 的内存为 40GB。

为了评估 ANNA 的性能，我们实现了一个 ANNA 的周期级模拟器，并实现了内存交叉访问模块用于模拟分布式架构中的异地内存的访问。使用的配置为 8 个 ANNA 的分布式架构，每个 ANNA 有 128 个 CU 单元和 36 个 SCM 模块。

ANNA 模拟器的代码结构如下:

(a) ANNA 的三个主要模块及其子模块

- cluster_codebook_processing_module.hpp
- encoded_vector_fetch_module.hpp
- similarity_computation_module.hpp
- memory_reader.hpp

(b) 内存接口

- memory_switch.hpp
- multi_memory_interface.hpp
- similarity_computation_module.hpp
- memory_reader.hpp

(c) 控制和配置组件

- config.hpp,config.cpp
- multi_anna.hpp
- utils.hpp
- test.cpp

对于向量数据库，使用 SIFT1B 和 DEEP1B 这两个广泛使用的十亿级的向量数据库作为数据集。

SIFT1B 和 DEEP1B 是两个常用的近似最近邻搜索的数据集，用于评估和比较不同算法和实现方法在大规模数据集上的性能。以下对这两个数据集进行简要介绍：SIFT1B 数据集是由 10 亿个 SIFT 特征向量组成的数据集，每个向量有 128 维。它是大规模图像检索中最常用的数据集之一。该数据集由标准的 SIFT 特征提取算法生成。SIFT1B 数据集被广泛用于评估近似最近邻搜索算法的准确性和效率。

DEEP1B 数据集是一个包含 10 亿个深度学习特征向量的大规模数据集. 这些特征向量是通过在深度神经网络 (如 CNN) 上进行特征提取得到的。DEEP1B 数据集用于评估近似最近邻搜索算法在处理深度学习特征向量时的性能。

4.3 界面分析与使用说明

在 config.hpp 中可以设置数据集和结果的存储路径，并设置 ANNS 算法的运行参数，主要参数如图 4 所示。

```
const std::string database_path = "./resource/union/query/";
const std::string result_path = "./result/";
extern std::string output_path;

const uint64_t W = 128; // also known as nprobe
const uint64_t MEMORY_SIZE = 8 * (1ull << 30);
const uint64_t CLUSTER_NUM = 16384;
const uint64_t METADATA_LEN = 64;

const uint64_t QUERY_NUM = 1024;
```

图 4. 配置示意

完成配置后，使用 CMAKE 编译运行即可。result 目录下会生成仿真运行的数据，output 目录下会生成运行过程的记录信息。

4.4 创新点

原文认为使用多个 ANNA 加速器可以有效提升性能，但并未详细介绍如何设计多个 ANNA 的协同工作流程和共享内存访问。我们设计了外部总线将多个 ANNA 加速器连接，控制其协同工作。为了避免内存总线成为性能瓶颈，我们为每个 ANNA 分配了本地内存，并将所有的内存合并编址。这允许通过统一的地址空间访问 ANNA 的本地内存和其他 ANNA 上的异地内存。

这样的设计增强了 ANNA 的可扩展性，避免单一的内存总线成为性能瓶颈。

5 实验结果分析

如图 5 所示，平均来看，ANNA 的处理速度是 FaissGPU 的 7 倍，是 FaissCPU 的 120 倍。这体现了 ANNA 专用架构的高效性。

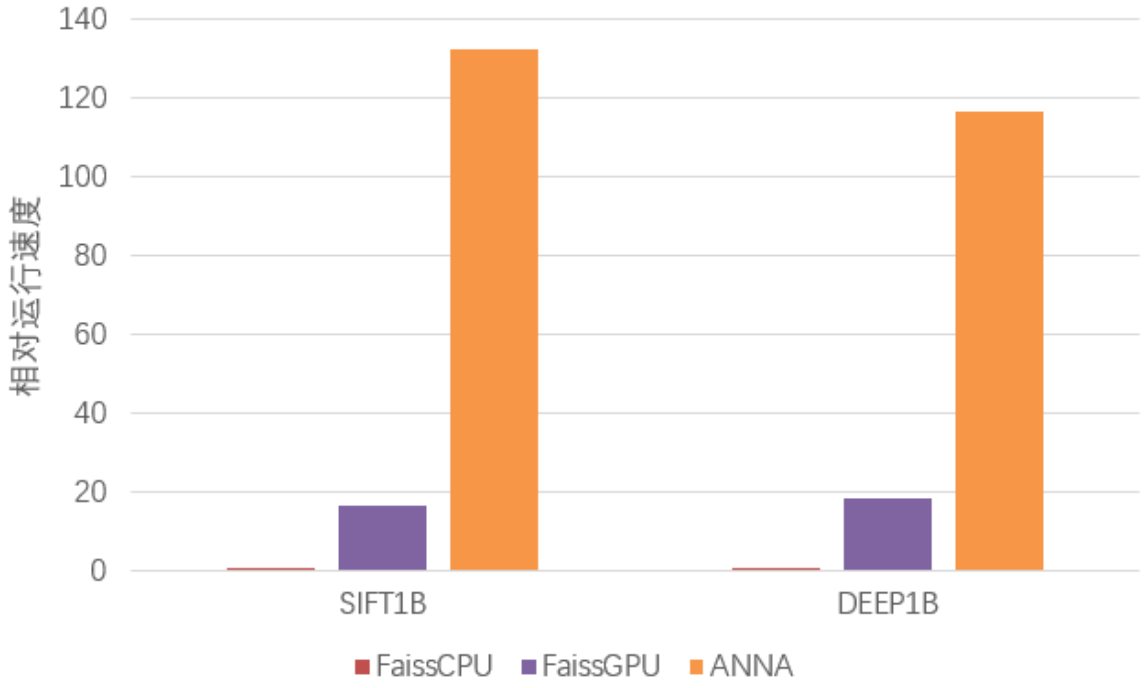

图 5. 实验结果示意

6 总结与展望

我们复现了用于大规模向量数据库中近似最近邻搜索的专用硬件架构 ANNA。我们使用周期级的模拟器复现了 ANNA 的各个模块及访存优化技术，并使用该模拟器仿真评估了 ANNA 的性能。我们还设计和实现了多个 ANNA 的协同工作流程和共享内存访问机制，提高了 ANNA 的可扩展性。未来，我们将对该硬件架构的能耗、芯片面积等指标进行评估和分析，并探索新的改进方法和技术。

参考文献

[1] Ameer M.S. Abdelhadi, Christos-Savvas Bouganis, and George A. Constantinides. Accelerated approximate nearest neighbors search through hierarchical product quantization. In *2019 International Conference on Field-Programmable Technology (ICFPT)*, pages 90–98, 2019.

[2] Dimitrios Danopoulos, Christoforos Kachris, and Dimitrios Soudris. Fpga acceleration of approximate knn indexing on high- dimensional vectors. In *2019 14th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 59–65, 2019.

[3] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, page 253–262. Association for Computing Machinery, 2004.

[4] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized product quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(4):744–755, 2014.

[5] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3887–3896. PMLR, 13–18 Jul 2020.

[6] Ben Harwood and Tom Drummond. Fanng: Fast approximate nearest neighbour graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5713–5722, 2016.

[7] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data (TBD)*, 7(3):535–547, 2021.

[8] Benjamin Klein and Lior Wolf. End-to-end supervised product quantization for image search and retrieval. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5036–5045, 2019.

[9] Yejin Lee, Hyunji Choi, Sunhong Min, Hyunseung Lee, Sangwon Beak, Dawoon Jeong, Jae W. Lee, and Tae Jun Ham. ANNA: Specialized Architecture for Approximate Nearest Neighbor Search. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 169–183, 2022.

[10] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *Computer Architecture Letters (CAL)*, 19(2):106–109, 2020.

[11] Yu A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[12] Marius Muja and David G. Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[13] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[14] Hao Sun, Xinzhe Liu, Qi Deng, Weixiong Jiang, Shaobo Luo, and Yajun Ha. Efficient fpga implementation of k-nearest-neighbor search algorithm for 3d lidar localization and mapping in smart vehicles. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(9):1644–1648, 2020.

[15] Tiancheng Xu, Boyuan Tian, and Yuhao Zhu. Tigris: Architecture and algorithms for 3d perception in point clouds. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '52, page 629–642. Association for Computing Machinery, 2019.

[16] Jialiang Zhang, Jing Li, and Soroosh Khoram. Efficient large-scale approximate nearest neighbor search on opencl fpga. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4924–4932, 2018.