并行进阶要求选题–ANNS

1 选题背景

最近邻搜索（NNS）的主要目标为在高维空间中识别与查询向量最接近的向量。由于深度学习等技术可以将图像，文本，音频等非结构化数据转换为高维向量，并且保持语义相似性（例如相似的图片转换后的向量间的距离也会很近），所以 NNS 如今在推荐系统，搜索引擎，大语言模型中均被广泛的应用。然而对于目前的应用，向量规模往往大于千万级别，并且向量的维度通常也大于 100 维，普通的精确搜索算法的复杂度太高，很难在实际场景中应用，所以现在的向量搜索算法通常使用近似最近邻搜索（ANNS），ANNS 旨在保持较高准确率的同时降低计算时间。

2 问题定义

给定一个 d 维向量数据集合 $X = \{x_1, x_2, x_3, ...x_N\} \subset \mathbb{R}^d$, 用 $\delta(x, y)$ 表示向量 x 和向量 y 的距离。给定一个查询 $q \in \mathbb{R}^d$。kNNS 目的在于找到集合 S，S 满足 $|S| = k, S \subseteq X$ 且

$$\forall x \in S, \forall y \notin S, \delta(x, q) < \delta(y, q)$$

直观理解：集合 S 是 X 中距离 q 最近的前 k 个向量组成的集合。

而 ANNS 并不要求集合 S 中的 k 个向量都是真实最近的 k 个向量，只要求能达到一定的准确度即可。我们通常使用查询延时 (latency) 和 recall@k 来衡量 ANNS 算法的效率。其中 recall@k 定义为

$$recall@k = \frac{KNN(q) \cap KANN(q)}{k}$$

其中 $KNN(q)$ 表示查询的精确答案集合，$KANN(q)$ 表示使用 ANNS 找到的答案集合。对于 ANNS 算法，我们在比较加速比时通常选择比较达到某一精度时不同算法的延时（例如达到 recall@100 = 0.95 时的延时）。

3 算法简介

ANNS 算法的优化可以分为两个方面

1. 对距离计算速度的优化，即加快 $\delta(x, y)$ 的计算速度，
2. 减少需要访问点的数量，普通的算法对于每一个查询，需要遍历 X 中的每一个点。而一个好的 ANNS 算法会尽可能减少访问点的数量。

另一点需要注意的是，ANNS 算法通常分为索引构建（离线）和查询（在线）。我们一般重点关注查询的性能，索引构建的性能通常放在第二位考虑。所以在本学期的并行实验中，我们重点对查询进行优化。（当然如果你对索引构建性能有好的优化算法，同样可以写在报告中）

3.1 SIMD

在 SIMD 实验中，我们重点关注对距离计算速度的优化。一般来说，$\delta(x,y)$ 使用欧几里得距离和内积距离。其中欧几里得距离定义为：

$$\delta(x,y)=\sum_{i=1}^{d}(x_i-y_i)^2$$

内积距离定义为：

$$\delta(x,y)=1-\sum_{i=1}^{d}x_i*y_i$$

朴素的 SIMD 优化是对上述的计算公式用 SIMD 指令进行加速。

一个更好的对距离计算进行优化的算法为 Vector Quantization (VQ)。其中 VQ 又分为 Scalar Quantization (SQ) 和 Product Quantization (PQ)。

3.1.1 SQ

由于原向量一般每一维度都用一个 32bit 的浮点数组存储，假如我们使用 128bit 的 SIMD 寄存器加速计算，最多只能一次算 4 个浮点数。但是 SQ 可以将向量的每一个维度量化为 8bit 的无符号整型，此时我们再用 128bit 的寄存器加速计算时，可以一次算 16 个 uint8。一个 SQ 的简易参考资料为https://zhuanlan.zhihu.com/p/719012156

3.1.2 PQ

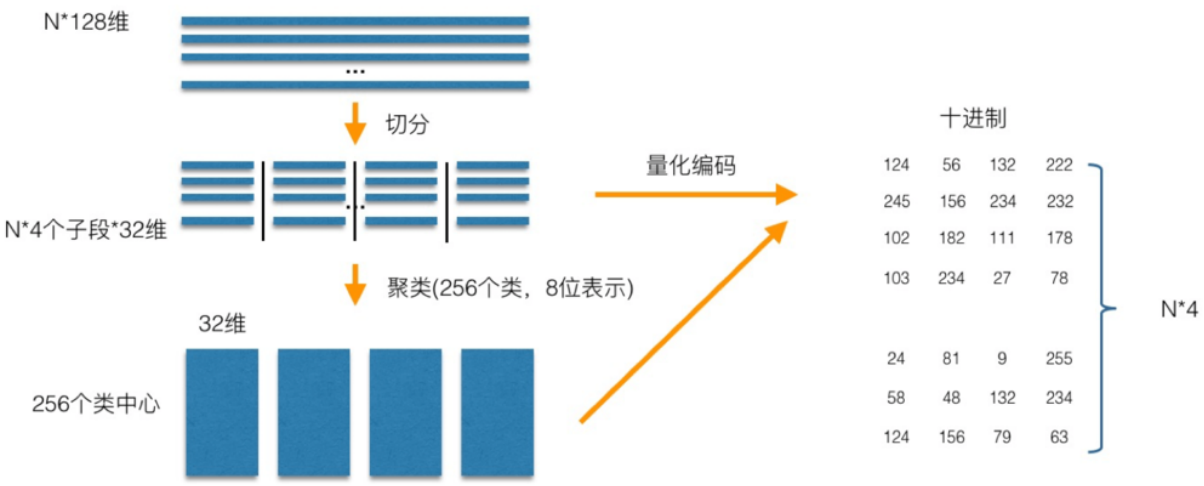

图 3.1: PQ 示例

以 N 条 128 维的向量为例，每条向量由 128 个浮点数组成，$(x_0,x_1,\ldots,x_{126},x_{127})$；首先对所有向量进行切分，如图中切分为 4 段所示，即每条向量的 $(x_0,\ldots,x_{31})$ 为第一段，$(x_{32},\ldots,x_{63})$ 为第二段，

$(x_{64},...,x_{95})$ 为第三段，$(x_{96},...,x_{127})$ 为第四段；第二步，对每一个子段进行聚类，将四个子段当作四个独立的子空间，分别对切片得到的 N 个 $(x_0,...,x_{31})$、N 个 $(x_{32},...,x_{63})$、N 个 $(x_{64},...,x_{95})$、N 个 $(x_{96},...,x_{127})$ 进行聚类，每一段聚类得到 256 个类中心，将总共 256×4=1024 个类中心进行保存并按顺序标号；第三步，对原始向量进行量化编码，将原始向量的四个段分别用距离最近的聚类中心的序号进行表示。以图中第一条向量为例，该向量的第一段 $(x_0,...,x_{31})$ 距离第一段的第 124 个类中心最近，因此向量的第一段 32 个浮点数替换为一个整数 124，第二、三、四段以此类推，最终一条由 128 维浮点数组成的向量压缩成了 4 个整数。

当计算查询向量与原始向量距离时，对查询向量进行同样的切分和量化编码，假设查询向量编码后得到 (25,82,10,254)，计算与第一条向量的距离为例，第一段中只需要计算第 25 个类中心和第 124 个类中心的距离，第二段计算第 56 个类中心和第 82 个类中心的距离，第三、四段同理，最后将四段的距离相加的结果作为查询向量与原始向量之间的距离的近似值。虽然原始向量和查询向量都是通过距离最近的类中心替代进行计算，得到的距离与实际距离有一部分偏差，但大大降低了查询的时间。

3.1.3 更多的探索

这一节中提到的技术难度较大，学有余力的同学可以进行自主探索。

实际上，单纯的使用上述的算法仍然不能满足如今大规模检索的性能需求，目前的 PQ 通常与一种叫做 FastScan 的技巧结合使用，FastScan 首次在论文Cache locality is not enough: high-performance nearest neighbor search with product quantization fast scan中被提出，同学们可以参考该论文和FastScan实现细节进行探索。

同时，同学们可以探索更多的 VQ 算法，例如

1. Optimized Product Quantization
2. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search

3.2 pthread/openmp

在后续的实验中，我们重点关注如何减少需要访问点的数量。一种朴素的方法是将 X 中的向量划分到各个线程中去，这样每个线程只需要访问一部分向量。

3.2.1 IVF 和 IVF-PQ

IVF(lnverted File) 在预处理阶段对 X 中的向量进行 KMeans 聚类，对于每个查询，检索距离最近的 m 个聚类蔟（这一步中，用聚类中心到查询的距离来代替簇中的其他点到查询的距离），遍历这 m 个聚类蔟下所有候选，计算得出距离最近的 topk 候选，在遍历 m 个簇时我们可以使用并行来加速。我们一般可以通过调整 m 来提高 recall@k，但同时也会提高 latency，这是一个 latency 和 recall@k 的 tradeoff，**在实验报告中需要有图表体现不同算法，不同线程数的 latency-recall@k tradeoff。**

IVF-PQ 即将 IVF 和 PQ 结合起来，可以参考https://zhuanlan.zhihu.com/p/378725270。

3.2.2 更多探索：图索引

这一节中提到的技术难度较大，学有余力的同学可以进行自主探索。

目前真正主流的 ANNS 算法是基于图的方法，其在各种场景下都取得了远高于其他算法的性能，目前在工业界中应用最广的图索引为 HNSW，原论文为Efficient and robust approximate nearest neighbor

search using Hierarchical Navigable Small World graphs。同学们在探索图索引算法时，可以直接使用一个 HNSW 的开源库hnswlib并在其基础上进行开发。

图索引的并行优化难度非常大，但我们仍然可以从以下几个方面进行考虑：

1. 搜索时将图中每个点的边划分到各个线程，每个线程负责搜索时访问一部分的边，可以探索边划分策略对性能的影响等。
2. 考虑每个线程搜索一部分点，再将各个线程搜索到的结果合并。此方法需要谨慎设计各线程的同步策略。

3.3 MPI

MPI 的整体优化思路与多线程大致相似，同学们可以自由探索更多的方法，例如用 MPI 实现 IVF+PQ，IVF+HNSW，HNSW+HNSW（即两层 HNSW，第一层 HNSW 中的每个点代表一个小的 HNSW，查询时进行嵌套的搜索）等。

3.4 GPU

GPU 优化通常以提高查询的吞吐率为主（即 Query 间并行），同学们可以对查询集或数据集进行划分，每个线程块负责一部分的查询任务或数据集任务。也可以考虑 GPU 特殊的架构特点，设计其他 GPU 并行算法。另一点是 ANNS 的计算也可以转换为矩阵乘法的计算，同学们可以自行进行相关公式的推导以及编写 GPU 加速代码来加速矩阵乘法。

3.5 期末报告

1. 在之前实验基础上探索更多的 ANNS 算法与更多的优化。
2. 将之前的实验和代码整理完善，形成一个完整的工作。
3. 对比不同并行环境下不同算法的性能。

4 代码框架说明

4.1 登录服务器

推荐使用 Vscode 的 Remote-SSH 连接服务器, 方便代码阅读和编写, 对 vim 熟悉的同学可以使用终端直接连接。

服务器分两层, 使用以下指令通过跳板服务器一键连接到 OpenEuler 集群主节点。

```
# 把 2222222 换成你的学号
# 连接指令
ssh -J s2222222@10.137.144.91:9001 s2222222@192.168.90.141
```

```
# Vscode ssh config 信息
Host s2222222
```

```
HostName 192.168.90.141
ProxyJump s2222222@10.137.144.91:9001
User s2222222
```

两层服务器的账号为 s + 学号, 密码同账号, 第一次登陆后需要修改密码。

4.2 提交测试

ann 相关代码位于 ann 目录下, 提交脚本为 test.sh, test.sh 脚本内容已加密, 不可修改只可执行, 内容包括提交 qsub 脚本 (qsub.sh), 保存提交记录 (已隐藏) 和输出 test.o 到终端中, 脚本使用如下:

```
# 第一个参数为实验序号，1 代表 SIMD，以此类推
# 第二个参数为申请节点数，SIMD 实验只能申请 1 个
# 下面的脚本即提交 SIMD 的测试，申请 1 个计算节点
bash test.sh 1 1
```

运行脚本后，会自动编译 main.cc，**只允许运行 test.sh，不允许直接运行程序，否则会导致你的成绩没有被正确记录**。

执行脚本会返回 test.e 和 test.o 两个文件, 分别对应 std::cerr 和 std::cout 的输出信息, 默认输出结果在 std::cout 中, 禁止删除、修改已有的输出结果, 但可以添加额外的输出结果。

如果需要文件输入输出，需要把文件放在 files 文件夹下, 输出位置指定在 files 文件夹下, 这一部分可参考代码中的 build_index 函数的文件输出方式。

test.sh 脚本已经进行加密，禁止尝试进行破解或者修改，否则你可能会有挂科风险。

4.3 数据集和代码框架说明

本次实验使用 DEEP100K 数据集进行测试, 其中数据集中的向量类型为图片, 维度固定为 96。读取数据集的代码已经在框架提供，如下所示:

```
auto test_query = LoadData<float>(data_path + "DEEP100K.query.fbin",
test_number, vecdim);
auto test_gt = LoadData<int>(data_path + "DEEP100K.gt.query.100k.top100.bin",
test_number, test_gt_d);
auto base = LoadData<float>(data_path + "DEEP100K.base.100k.fbin",
base_number, vecdim);
```

这段代码执行完成后,test_query,base 均为 float* 类型,base_number 表示数据点的个数,vecdim 为向量的维度，test_number 为用于测试的查询个数，test_gt 为查询的答案，用于测试 ANN 算法的准确度，大家并不需要关注（**严禁直接将 test_gt 的内容直接作为答案输出**）。base 中存储着数据点向量，每 96 个 float 为一组（即前 96 个 float 为第一个向量，之后 96 个 float 为第二个向量，.....）; test_query 中存储着查询向量，存储格式与 base 一致。

```
auto res = flat_search(base, test_query + i*vecdim, base_number, vecdim, k);
```

框架中已经为大家实现最平凡的串行算法，即对于每一个查询，暴力遍历所有数据点中的向量（上面代码中的 flat_search）。大家需要自己实现更优的算法，然后将这一段函数调用替换为自己实现的，但一定注意函数返回值要与示例一致。同时如果大家想要新增文件，必须采用.h 的形式增加（即 header-only），**否则无法正常进行测试**。

编译命令如下所示：

```
g++ main.cc -o main -O2 -fopenmp -lpthread -std=c++11
# 该编译选项仅供大家本地调试用，无法在服务器上修改，
# 如果你确实需要修改编译选项，可以和助教私聊并给出详细理由。
```

4.4 注意事项

- 本学期的并行实验中，会自动记录你每次的提交记录和结果，**所以如果你伪造数据会很容易被发现**。
- 本学期的并行实验从 SIMD 开始，会对代码进行互相查重（~~大家应该在算法导论课上体验过子~~），如果查重不通过可以联系助教进行申诉。
- 每次实验截止后，大家需要提交一份能够直接通过 test.sh 运行，且运行的算法是你实现的最复杂或最快的（即选择你工作中最具有代表性的）。助教会统一运行这份代码来检查你的工作量是否与报告中一致。
- 由于服务器空间有限且上课人数较多，建议大家定期清理自己目录下没有用的索引文件，防止服务器磁盘空间不足。
- 尽早开始你的任务，不要拖到 ddl 再赶，这时候服务器压力会非常大，**你可能会因为提交不上任务而错过 ddl**。
- 服务器的波动可能较大，大家可以在不同的时间段多测试几次然后选择最优的数据。
- 大家在服务器上提交测试前，**需要保证自己代码已经在本地 arm 虚拟环境下验证过正确性**，大家可以直接将服务器上的代码拷贝到本地然后自己使用上面提供的编译命令编译后运行。如果你提交的代码里面有死循环，可能会导致服务器内存不足或者磁盘没有剩余空间，这时服务器会直接崩溃，**如果你多次因为很低级的错误导致服务器崩溃，可能有扣分的风险。如果你恶意对服务器进行破坏，可能有挂科的风险**。
- 推荐大家定期在本地备份自己的代码，避免因为服务器崩溃等原因导致自己代码丢失。