HNSW + 并行编程项目报告

韦东良 522031910516

2024 年 4 月 15 日

1 背景介绍

本次 Lab 的主要内容是实现一个名为 Hierarchical-Navigable-Small-World（HNSW）的图相关数据结构及其算法，并进行并行优化。HNSW 是一种基于图的 ANN 索引，用于查找与输入向量相对接近的向量。

HNSW 是基于 NSW 进行优化的。NSW 将数据库中的向量与接近的向量相连，形成一个连通图。查询过程从连通图上的起始节点开始，不断跳到更靠近目标节点的邻居节点，直到无法再靠近目标节点为止，此时的终点节点即为目标节点的临近节点。

在 NSW 基础上，HNSW 借鉴了跳表的思想，采用分级的方式存储特征向量。导航过程从入口节点开始，在较高层级尽可能向目标节点靠近。如果无法继续靠近，则下降到下一层级的相同节点，直到最终下降到最底层并完成查询。

2 系统实现

我的 HNSW 的实现基本是按照 Lab 文档中给出的伪代码来实现的。遇到的难点是 Node 节点这一数据结构的设计，除了存储 label 和向量坐标 data 之外，还存了 std::unordered_map <int, std::unordered_set<Node*> > 类型的 neighbors 这一变量，即与这一节点在各层（int 表示层数）相连的邻居节点。印象深刻的细节是在第一次跑正确性测试之后 debug，发现问题出在把 neighbors[lc][e].add(q) 实现成了 neighbors[lc][e] = {q}，改正之后第二次跑正确性测试就顺利通过了。

我的并行查询的实现是 lock-free 的。我使用 C++ 的 thread 库来创建多个线程，用 vector<thread> threads 存储线程对象。我为每一个查询请求创建一个新的线程，添加到 threads；在这些线程中调用 hnsw.query 函数，彼此独立地处理查询；使用 vector <vector<int> > test_gnd_l (gnd_n_vec) 数组，将第 i 个线程得到的查询结果 test_gnd 存入 test_gnd_l[i]（之所以这样做，是因为如果用 test_gnd_l.push(test_gnd)，为了保证各个线程的 push 顺序不能乱，就必须加锁）。最后使用 join() 函数来等待所有线程完成。

3 测试：参数 M 的影响

3.1 测试配置

测试对象：不同 M 及 M_max 值下的 HNSW 实现（无并行优化）

工作负载：测试使用预设好的参数设置（M，M_max，efConstruction = efSearch = 100），数据集使用 siftsmall，向量维度为 128，需要插入 10000 条向量，执行 100 个查询请求。测试主函数首先读取输入文件，构建 HNSW 索引，执行查询操作，最后与通过暴力搜索得出的结果进行比较，记录召回率与单次查询时延。

系统配置：Ubuntu 20.04 系统

机器配置：VMware 虚拟机，内存：8GB, 硬盘（存储空间上限）：60GB，处理器数量：2

3.2 测试结果

表 1: 参数 M 的影响

M=M_max 的值	10	20	30	40	50
召回率	90.4%	97.7%	99.1%	99.4%	99.4%
单次查询时延（单位：ms）	2.9	5.2	5.8	6.7	6.1

3.3 结果分析

观察测试结果可以发现，随着 M 值的增大，召回率先是随之上升，最后稳定在一个固定值（99.4%）。原因可能是当 M 值增大，图中每个节点都会与更多其他节点建立连接，图上的有效信息增多，能更精确地逼近目标节点，减小了导航路径恰好避开最接近节点的可能性。但这样的效果是有限的，局部最优无法完全等同于全局最优，M 达到一定值后，导航路径无法更加优化，因此召回率趋于稳定，不能无限接近 100%。

随着 M 值的增大，单次查询时延先是随之增大，在 $M = 40$ 处达到最大，在 $M > 40$ 之后又有所减小。原因可能是 M 增大产生两种效果：（1）由于能更精确地逼近目标节点，导航路径上的节点数会减少，加快查询。（2）每个节点的邻居数增加，从一个节点挑选下一个要跳到的邻居节点的过程耗时增加，减慢查询。在 $M < 40$ 的时候，（2）减慢的效果强于（1）加快的效果，单次查询时延增大，在 $M > 40$ 的时候，（1）加快的效果强于（2）减慢的效果，单次查询时延减小。

4 测试：性能测试

4.1 测试配置

测试对象：串行查询下的 HNSW 与并行优化下的 HNSW

工作负载：测试使用预设好的参数设置（M = M_max = 30，efConstruction = efSearch = 100），数据集和测试集与上一节“参数 M 的影响”测试相同。测试函数分为串行查询和并行查询两部分，每个部分都读取输入文件，构建 HNSW 索引，执行查询操作，最后与通过暴力搜索得出的结果进行比较，记录召回率与单次查询时延。

系统配置与机器配置：与上一节“参数 M 的影响”测试相同

4.2 测试结果

表 2: 性能测试

操作	插入	串行查询	并行查询
单次操作时延（单位：ms）	4.5	6.0	2.8

4.3 结果分析

操作性能的强弱排序为：并行查询 > 插入 > 串行查询。并行查询性能明显优于串行查询，因为并行优化下能做到多个线程同时处理不同的查询请求，且相互不冲突，而单线程只能同时处理一个查询请求。

5 结论

本次 project 中，我掌握并实现了 HNSW，对查询请求进行了并行优化，进行了关于参数 M 的影响的测试与性能测试。综合来看：随着 M 值的增大，召回率先是随之上升，最后稳定在一个固定值。随着 M 值的增大，单次查询时延先是随之增大，在 M = 40 处达到最大，在 M > 40 之后又有所减小。操作性能的强弱排序为：并行查询 > 插入 > 串行查询。并行查询性能明显优于串行查询。