
南開大學
Nankai University

计算机学院

并行程序设计实验报告

# MPI 编程实验 ANN

姓名：杨宇翔

学号：2312506

专业：计算机科学与技术

2025 年 6 月 14 日

# 目录

# 1 前言

我目前实现了的 ANN 算法有 sq、pq 和 ivfpq，其中 ivfpq 在串行下性能最好。但是在第三次的 openmp 并行实验中，发现 sq 算法并行后的提升幅度是最大的，在保持 0.99 的高 recall 下延时只有三位数（$\mu s$），和 ivfpq 只相差了 345.24$\mu s$。所以这次实验除了 ivfpq 外，还要测试 sq 算法的并行效果。

# 2 算法设计

## 2.1 SQ

SQ 的主要算法就是直接遍历整个压缩后的 base，计算查询向量与每一个向量的距离，然后排序。所以并行化的思路就是将 base 平均划分给各个进程，每个进程分别计算查询向量与自己的 local_base 的距离并维护自己的 rerank，最后主进程 (rerank 0) 负责归约所有进程的结果，得到最终的前 rerank 名的查询结果。

由于多进程已经划分了任务，多线程就只能在距离计算上发挥作用了。我们可以在各个进程遍历自己的 local_base 时加上多线程来加速计算速度。SIMD 则可以用在向量距离函数里，用于加速浮点数乘法的运算。

具体来说，在每个查询向量的查询中，主进程会将查询向量广播给其他进程，此时所有进程包括主进程自己开始计算自己的前 rerank 结果。非主进程在计算完后会调用 MPI_Send 向主进程发送计算结果，主进程则会在自己的计算完成后循环 MPI_Recv 直至收齐所有计算结果。非主进程在发送完计算结果后会进入下一次查询向量广播的等待中，直至主进程的归约、重排等后续工作结束后，进入下一次循环开始广播查询向量为止。

虽然每一个查询向量是独立的，实际上并不需要等待主进程广播，其他进程可以提前把所有查询向量的结果都计算出来，最后再分批发送给主进程。但考虑到平均延时的计时是针对每个单独的查询向量而非批量的查询向量，实验的重点是加速单次查询的速度，所以为了计时的准确和公平，非主进程需要等待主进程完成一次完整的查询任务后才能进入下一个查询任务。

## 2.2 IVF-PQ

IVF-PQ 的算法流程是先确定距离查询向量最近的 m 个粗簇，然后遍历这 m 个粗簇下的所有向量。每个粗簇的计算是独立的，所以我们可以用 mpi 多进程来分配遍历粗簇的任务。每个进程维护自己的 rerank，最后主进程归约即可。

和 SQ 一样，多线程用于加速各个进程遍历粗簇下的向量，SIMD 用于加速向量距离计算。

具体来说，主进程先找出相距最近的 m 个粗簇，然后把粗簇索引广播给其他进程，各个进程就开始计算自己簇中前 rerank 的结果，最后主进程归约。循环处理和 SQ 相同，等待主进程结束一次完整的查询后再进行下一个查询。

# 3 实验分析

## 3.1 SQ

```
average recall: 0.993917
average latency (us): 541.308

Authorized users only. All activities may be monitored and reported.
```

图 3.1: 8 进程无 OpenMP 加速

上图是 8 进程无 OpenMP 加速下的实验结果截图，rerank 设置为 200。先说结论，这是性能相对最好的结果。下面是不同参数的实验讨论。

### 3.1.1 进程数对比

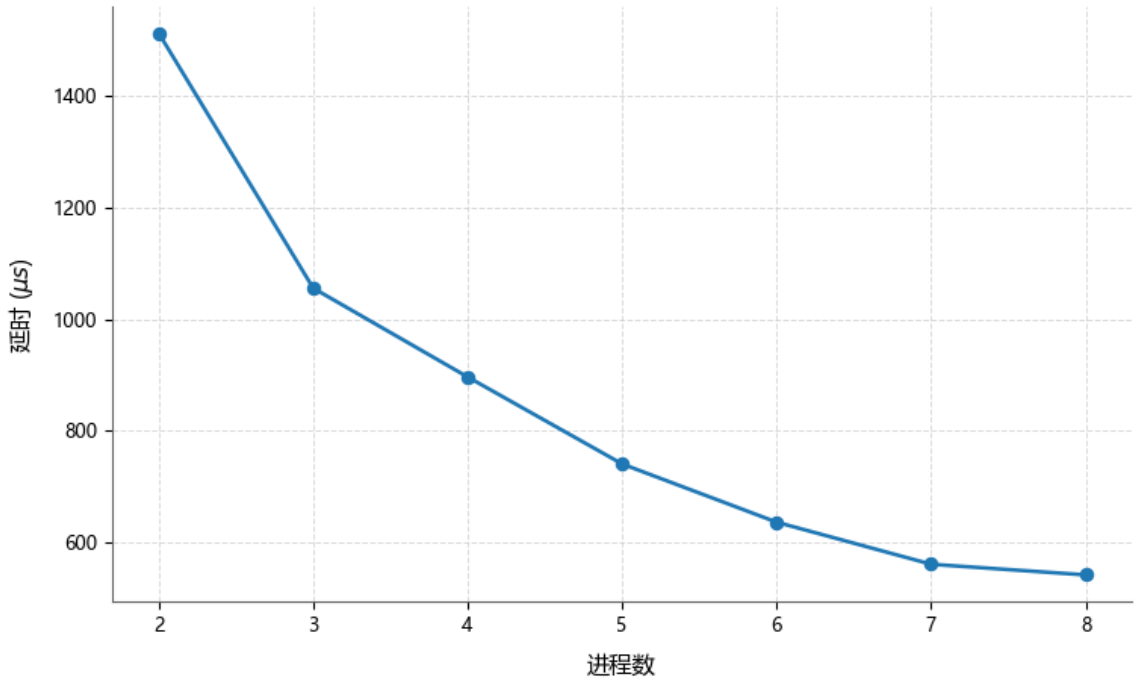


图 3.2: SQ 进程数性能对比

上图是 SQ 在**没有使用 OpenMP** 下的的延时随进程数变化的图像，rerank 设置为 200，recall=0.993917。

从图中可以看到延时随着进程数的增加显著减少，虽然在 8 进程时下降趋势削弱，但如果继续增大应该可以进一步降低延时，如果硬件支持的话。之所以不使用 OpenMP，一是要控制变量，这里只讨论进程数对性能的影响；二是因为多线程的效果非常差，具体实验接下来会讲。

### 3.1.2 OpenMP 线程数对比

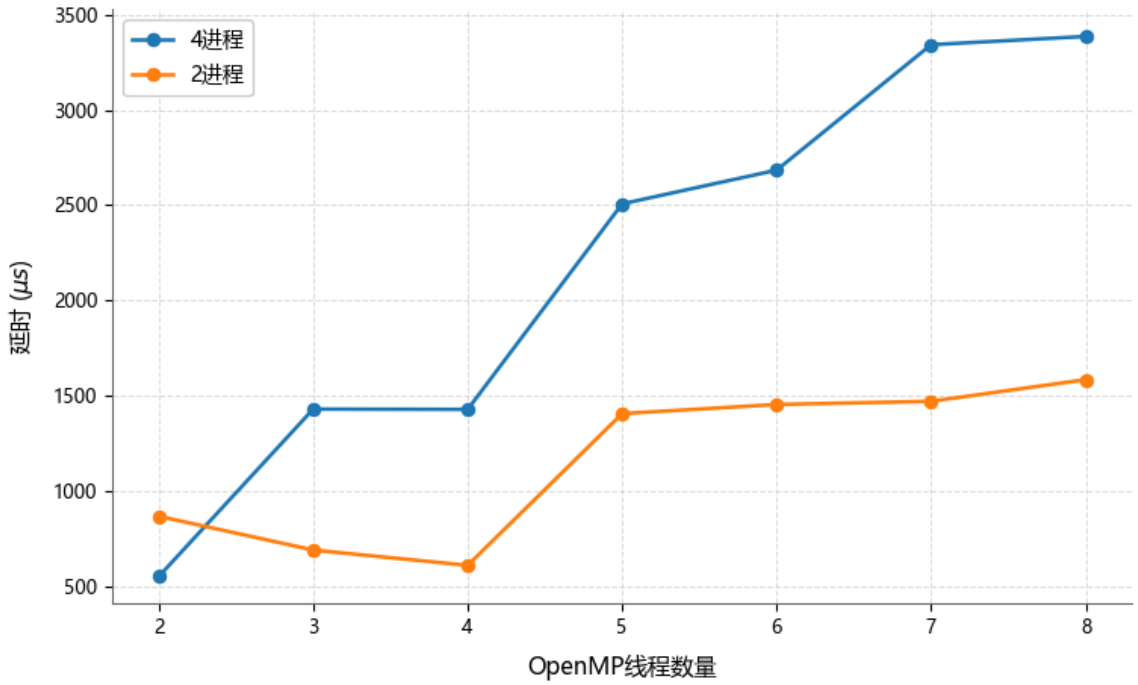


图 3.3: MPI 下不同 OpenMP 线程数性能对比

上图是 SQ 在分别在 2 进程和 4 进程下的不同 OpenMP 线程数对性能的影响，rerank 设置为 200，recall=0.993917。

从趋势上看，可以看到 4 进程的延时随线程数增多而越来越高，而 2 线程的延时则是在线程数等于 4 时出现反弹。而如果我们进一步观察，结合 8 进程无 OpenMP 时的数据，就会发现，8 进程无多线程、4 进程 2 线程、2 进程 4 线程，这三次实验的延时是几乎一样的。同样的规律在图上还表现在 2 进程的 5、6、7、8 线程的延时和 4 进程的 3、4 线程的延时几乎一样上。然后通过多次实验我们就能发现：**进程数乘以线程数等于 8 时性能最好**。这个规律同样适用于后面的 IVF-PQ 算法。

尽管进程数和线程数乘积相同下，不同组合的的延时都差不多，但多次实验后观察发现，相同乘积下，进程数越高性能越好，直接丢弃 OpenMP 是最好的。关于这一点，如果不考虑硬件资源的问题，我认为是多线程维护最大堆的开销较大导致的。多线程计算查询向量与候选向量的距离时，每个线程都需要维护自己的堆，最后归约时需要操作同一个全局的最大堆，这个过程必须上锁，就会产生开销。原本在多线程的实验中，最大堆的开销被候选向量，即需要遍历 base 的开销掩盖了，但在多进程里已经减少了 base 大小的情况下被放大了，就导致多线程的管理开销和内存竞争暴露出来。

虽然但是，从 2 进程的延时曲线上可以看到，4 线程前多线程还有一定的优化效果，但超过 4 线程后延时突然反弹，此时又恰好是进程数和线程数乘积为 8 的时候，我觉得可以合理怀疑是核心数之类的限制导致的延时飙升。

## 3.2 IVF-PQ

```
成功读入inverted_kmeans.bin!
成功读入ivf_centers.bin!
成功读入ivf_base.bin!
成功读入ivfpq_kmeans_base.bin!
成功读入ivfpq_kmeans_centers.bin!
average recall: 0.904752
average latency (us): 274.152

Authorized users only. All activities may be monitored and reported.
```

图 3.4: 最终性能

上图是经过参数调整后的最终性能，进程数为 8，无 OpenMP 多线程，使用 SIMD 加速，粗簇数量 m=16，rerank=320。接下来是各个参数的实验讨论。

### 3.2.1 进程数对比

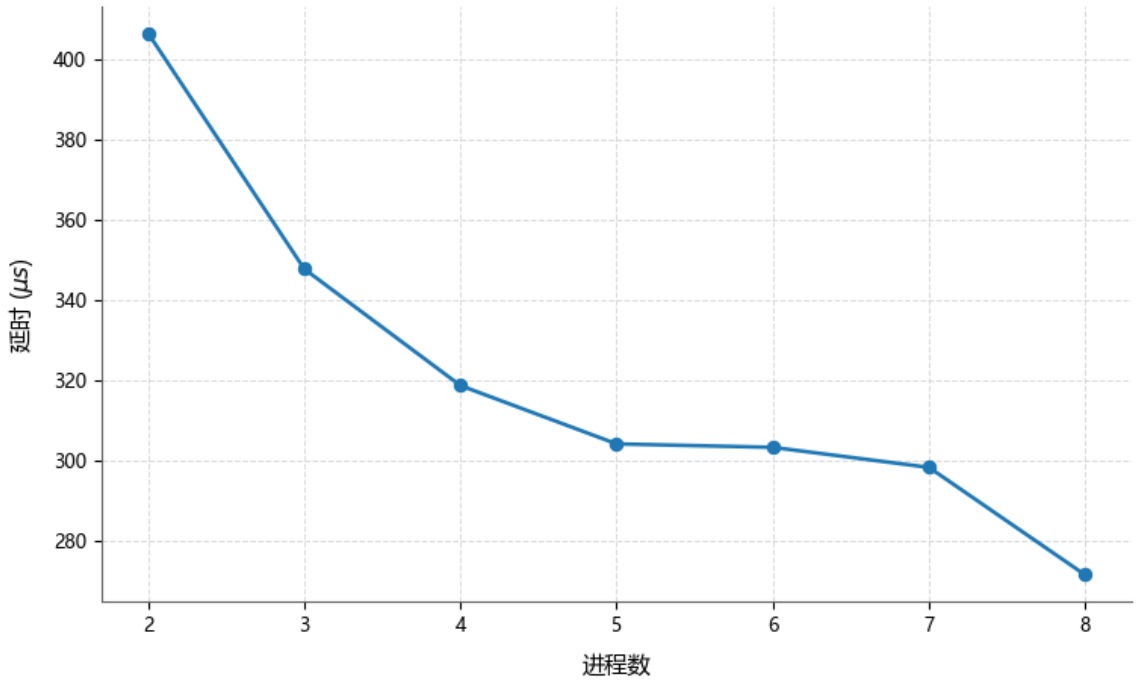


图 3.5: IVF-PQ 进程数性能对比

上图是无 OpenMP 多线程，使用 SIMD 加速，粗簇数量 m=16，rerank=320 下的延时曲线。可以看到延时随着进程数的增多有明显的下降。曲线出现拐弯主要是因为进程数除不尽，各个进程任务分配不均导致的。

### 3.2.2 OpenMP 线程数对比

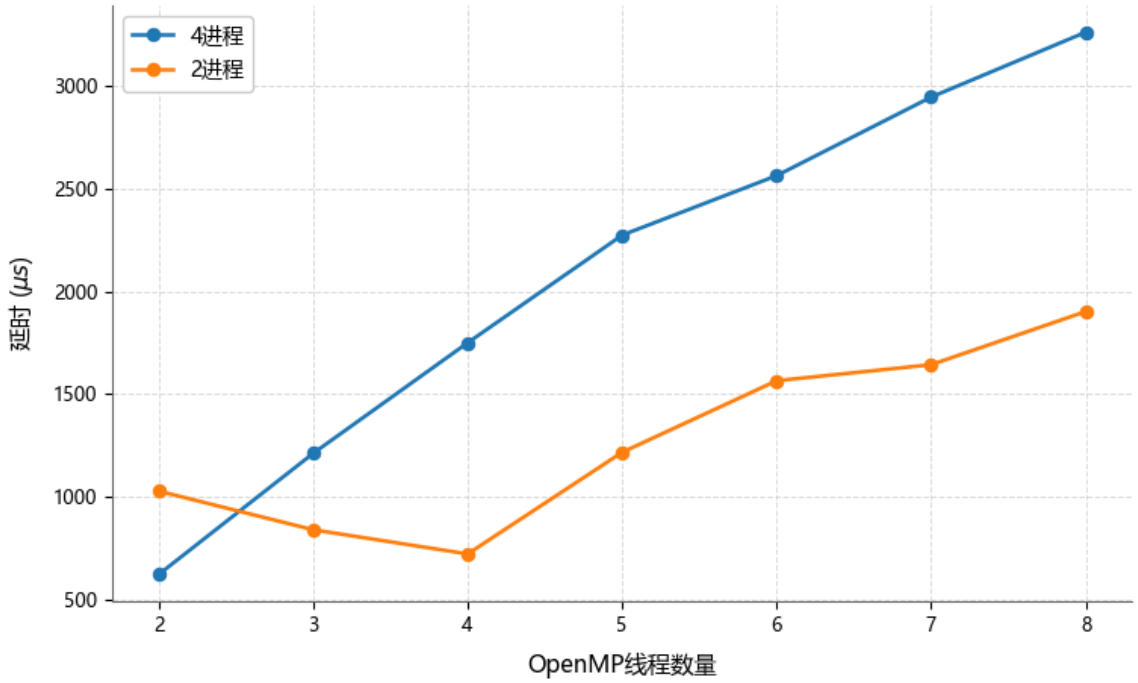


图 3.6: MPI 下不同 OpenMP 线程数性能对比

上图是 OpenMP 在 IVF-PQ 算法上的表现，使用 SIMD 加速，粗簇数量 m=16，rerank=320。在这张图上我们能发现和 SQ 实验一样的结论，当进程数和线程数的乘积为 4 时，性能是最高的，2 进程在 4 线程时也同样发生了反弹，这和我们的预期是一致的。

如果我们观察它们的最低点，就会发现在同样的粗簇数量和 rerank 设置下，它们的延时比最开始展示的 8 进程无多线程的延时要高得多。这是因为 IVF-PQ 算法已经通过粗簇筛选掉了大量的向量，剩下每个粗簇需要遍历的向量只有三位数，它的加速效果已经不能掩盖多线程的开销了。再加上多进程削弱了多线程的加速效果，就导致多线程完全是负优化，延时疯狂往上涨。

### 3.2.3 粗簇数量 m 对比

表 1: 粗簇数量 m 对性能的影响

| 粗簇数量 m | rerank | recall | latency($\mu s$) |
|---|---|---|---|
| 8 | 600 | 0.904084 | 358.758 |
| 16 | 320 | 0.904752 | **277.315** |
| 24 | 300 | 0.903835 | 304.386 |
| 32 | 300 | 0.905335 | 334.473 |

上表是 8 进程，无 OpenMP 多线程，使用 SIMD 加速后的不同粗簇数量下的性能对比。我通过调整 rerank 的值来保持 recall 大致相同，以此对比不同粗簇数量下延时的差距。

可以看到当粗簇数量为 16 时的性能最好，这是很好解释的。由于我们使用了多进程来分配各个粗簇，原本增加候选簇带来的距离计算开销就被大幅度削弱，此时就可以在增加粗簇数量的同时减少 rerank，减轻维护最大堆的开销。但当粗簇数量上升时，效果出现了反弹，观察粗簇数量为 24 和 32 的 rerank 和 recall，在相同 rerank 的情况下，后者多了 8 个候选簇 recall 却只上升了 0.0015。这显然是因为我们需要的前 rerank 向量已经几乎全部包含在了前 16 个粗簇里，后面继续增加粗簇数量意义不大，反而会因为粗簇带来的额外计算开销造成 latency 上涨。

# 4 总结

表 2: 各版本性能

| **指标** | **SQ OpenMP** | **SQ MPI** | **IVF-PQ OpenMP** | **IVF-PQ MPI** |
|---|---|---|---|---|
| recall@10 | 0.9938 | **0.993917** | 0.903254 | 0.904752 |
| latency ($\mu s$) | 791.127 | 541.308 | 445.888 | **274.152** |

上表是这次多进程 MPI 实验和上一次 OpenMP 实验的四个并行版本的最终性能对比。

可以看到通过多进程，我们的性能又一次得到了巨大的提升。不得不提的是，SQ 算法虽然简单，但并行的效果是真的好，维持极高的 recall 的同时延时也很低。虽然 IVF-PQ 最终还是性能最高的，但可以预见的是，在下一次的 GPU 编程中，分支预测少、指令简单、访存连续、任务划分均匀的 SQ 算法在性能上会得到质的飞跃，而 IVF-PQ 将受限于它复杂的预处理步骤得不到太好的优化。

Git 项目链接：https://github.com/SheepSpaceFly/NKHW/tree/main/并行程序设计/MPI 多进程实验