南开大学

Nankai University

南　　开　　大　　学

计　算　机　学　院

并行程序设计实验报告

ANN 选题 GPU 并行编程

姓名：申健强

学号：2313119

专业：计算机科学与技术

指导教师：王刚

2025 年 7 月 2 日

目录

一、 前言

(一) 实验环境

- CPU：vCPU10 核 32GB 内存
- GPU RTX3090 显存 24GB
- IDE：Vscode
- 编译器：gcc11.4.0、Cuda-nvcc11.5
- 操作系统：Ubuntu22.04

(二) 实验内容概述

1. 问题定义

算法的背景和意义在前三次实验中已详尽介绍，在此处我们只需表述问题的定义。

- 给定数据集 $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^d$。
- 对于查询向量 $\mathbf{q} \in \mathbb{R}^d$，ANNS 的目标是找到集合 $S \subseteq X$，使得 S 中的向量与 $\mathbf{q}$ 足够接近，并满足一定的准确度要求。

准确度衡量：常用召回率（recall）评估 ANNS 的准确性：

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

其中 $KNN(\mathbf{q})$ 是精确的 k 个最近邻集合，$KANN(\mathbf{q})$ 是 ANNS 返回的 k 个近似最近邻集合。本实验中，我们关注在达到特定召回率（0.9）时的算法延迟。

2. ANNS 的优化方向

ANNS 的优化主要集中在以下两个方面：

1. **加速距离计算**：
   - 使用 SIMD 指令并行化距离计算，提升单次计算的效率。
   - 采用量化技术（如 PQ、OPQ、SQ）减少浮点运算量，降低计算复杂度。
2. **减少访问的数据点数量**：
   - 构建高效索引结构（如 HNSW、IVF）以快速定位候选集，减少需要计算的向量数量。
   - 使用混合方法（如 IVF+HNSW）结合快速粗筛和精确查找，在保证准确度的同时优化性能。

本次实验我们将重点利用 CUDA 编程使用 GPU 对 ANN 进行并行加速，所以尤其需要重视 CUDA 编程的基本模式：在 ANN 选题由于 GPU 很难对单个查询进行上述两个层面的加速，所以 GPU 加速通常考虑将多个查询作为一个 batch，然后对一个 batch 的查询共同处理（即查询间并行）。在本次实验中我们将首先通过核函数及其优化实现 ANN 的批次矩阵化并行。另一方面，我们也将考虑在 IVF 上如何进行查询间并行。我们可以将 IVF 的一个簇里的向量也写成一个矩阵，在计算一个簇中向量到查询距离时使用矩阵乘法。在本次实验中我们也将进行相关尝试。

(三) 实验代码

与前几次实验相同，本人实验的所有代码，实验结果，包括但不限于可视化和算法脚本都可以在https://github.com/sjq0098/Parallel_Programs.git中查看。

二、 算法加速原理

(一) batch 批次并行搜索

内积距离的计算实际上与矩阵的一行和一列相乘是一致的，所以 ANN 的距离计算实际可以转换为矩阵乘法进行计算。假设数据集 basedata 大小为 n，维度为 d，一个 batch 的查询数量为 m。那么可以将基数据集写成一个大小为 $n \times d$ 的矩阵，将查询写成一个大小为 $d \times m$ 的矩阵，相乘后可以得到一个大小为 $n \times m$ 的矩阵，最后在该矩阵的每一列中取距离最小的前 k 个点，即为该列对应查询的最近邻。具体公式如下：

$$B = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_n^\top \end{bmatrix} \in \mathbb{R}^{n\times d}, \quad Q = \begin{bmatrix} q_1, & q_2, & \dots, & q_m \end{bmatrix} \in \mathbb{R}^{d\times m},$$

$$S = BQ \in \mathbb{R}^{n\times m}, \quad \text{其中} S_{ij} = x_i^\top q_j.$$

对每一列 $j = 1, \dots, m$，取

$$\mathrm{NN}(q_j) = \arg\mathrm{topk}_{i=1,\dots,n} \{ S_{1j}, S_{2j}, \dots, S_{nj} \},$$

即为查询 q_j 的前 k 个最近邻索引。

(二) IVF 上的查询间并行与分组策略

为了在 IVF 中对一个 batch 的多个查询并行搜索，一个自然思路是对每个查询 q_j 先选出其前 n_{probe} 个粗排簇：

$$C_j = \{ c_{j,1}, c_{j,2}, \dots, c_{j,n_{\text{probe}}} \}, \quad j = 1, \dots, m.$$

直接对每个簇内部向量做矩阵乘法会造成重复计算。我们希望将 batch 内查询划分为 G 个组，使得同一组内查询的粗排簇集合重叠度高，从而减少实际参与乘法的簇总数。

- 定义分组函数

  $$g : \{1, \dots, m\} \longrightarrow \{1, \dots, G\},$$

  将查询索引 j 映射到组编号 $g(j)$。

- 对每个组 g，令组内查询索引集合为

  $$\mathcal{J}_g = \{ j \mid g(j) = g \}, \quad |\mathcal{J}_g| = m_g.$$

- 该组所需扫描的簇编号集合取并集

  $$U_g = \bigcup_{j\in\mathcal{J}_g} C_j, \quad u_g = |U_g|.$$

- 将这些簇中的所有向量拼成矩阵

$$B_g \in \mathbb{R}^{u_g \times d}, \quad Q_g \in \mathbb{R}^{d \times m_g},$$

然后一次性计算

$$S_g = B_g Q_g \in \mathbb{R}^{u_g \times m_g}.$$

- 最后，对每个查询 $j \in \mathcal{J}_g$，只在其对应子簇行索引子集 $C_j \subseteq U_g$ 中挑选 top-k 的结果即可。

这样，组 g 的计算量约为 $O(u_g m_g)$，总量为

$$\sum_{g=1}^{G} O(u_g m_g),$$

若分组合理，则 $\sum u_g m_g \ll \sum_j n_{\text{probe}} \cdot 1 = m n_{\text{probe}}$，从而显著减少无效乘法。

备注： 分组函数 $g(\cdot)$ 可通过对每个查询的簇集合 C_j 做相似度度量后聚类获得，也可使用启发式的“按簇编号排序后滑窗”方式快速构造，在后续我们将设计实验深入探究这个问题。

三、 算法实现及其优化

(一) batch 批次并行搜索

我们实现了两种基于 GPU 加速的批次并行向量相似性搜索方案，主要区别在于 TOP-K 选择阶段的计算位置。两种方案均使用 cuBLAS 进行矩阵乘法运算，并采用 CUDA 流实现异步数据传输。

1. GPU 矩阵运算 + CPU 选择 TOP-K

Algorithm 1 GPU 矩阵运算 + CPU TOP-K 选择算法

1: **输入：** 基向量集合 $B \in \mathbb{R}^{N \times D}$，查询向量集合 $Q \in \mathbb{R}^{B \times D}$，取顶值 k
2: **输出：** 每个查询的 TOP-K 结果
3: 在 GPU 上分配并传输数据：`d_base`, `d_query`, `d_distances`
4: 使用 cuBLAS 计算：

$$D \leftarrow Q \times B^{\top}$$

5: 在 GPU 上将内积转换为距离：

$$\texttt{dist}_{i,j} \leftarrow 1 - D_{i,j}$$

6: 将距离矩阵 `d_distances` 传回 CPU
7: **for** each query $i = 1, \ldots, B$ **in parallel do**
8: 在 CPU 上用优先队列执行 TOP-K
9: **end for**
10: 释放 GPU 资源并返回结果

2. GPU 矩阵运算 + GPU 选择 TOP-K

我们将这个算法简记为 GPU+GPU 算法，其实现如下：
其中 TOP—K 选择的算法如下：

Algorithm 2 GPU 矩阵运算 + GPU TOP-K 选择算法

输入： 基向量集合 $B \in \mathbb{R}^{N\times D}$，查询向量集合 $Q \in \mathbb{R}^{B\times D}$，取顶值 k

输出： 每个查询的 TOP-K 结果

在 GPU 上分配并传输数据：`d_base`, `d_query`, `d_distances`, `d_results`

使用 cuBLAS 计算：

$$D \leftarrow Q \times B^{\top}$$

在 GPU 上将内积转换为距离：

$$\texttt{dist}_{i,j} \leftarrow 1 - D_{i,j}$$

复制距离矩阵：`D_copy` = `distances`

在 GPU 上执行 TOP-K 选择

将结果传回 CPU 并返回

Algorithm 3 GPU TOP-K 选择（简化版）

输入： 距离矩阵 $D \in \mathbb{R}^{B\times N}$，取顶值 k

for each query $q = 1, \ldots, B$ **in parallel do**

  for $i = 1, \ldots, k$ **do**

    找到未选元素中最小距离 $\min\_dist$ 及其索引 $\min\_idx$

    将 $(\min\_dist, \min\_idx)$ 加入结果集

    标记 $D[q, \min\_idx] = +\infty$

  end for

end for

(二) IVF 上的查询间并行与分组策略

1. 查询间并行实现

查询间并行是 GPU IVF 索引的核心优化策略，通过将多个查询向量组织成批次进行并行处理，充分利用 GPU 的矩阵运算能力和内存带宽。其核心思想是将单个查询的串行处理转化为批量查询的并行计算，从而实现显著的性能提升。

Algorithm 4 GPU IVF 批量并行搜索算法

Input: 查询集合 $Q = \{q_1, q_2, \ldots, q_B\}$，IVF 索引参数 $(C, L, nprobe)$，返回结果数 k

Output: 每个查询的 Top-K 近邻结果集合 $R = \{R_1, R_2, \ldots, R_B\}$

Phase 1: 簇分配并行计算

将查询矩阵 $Q_{B\times d}$ 复制到 GPU 内存

计算查询-簇中心距离：

$$D_{B\times nlist} = Q \times C^{\top}$$

▷ kernel: `compute_query_centroid_distances`

for each 查询 q_i **in parallel do**

$clusters_i \leftarrow \mathrm{TopK}(D[i,:], nprobe)$ ▷ kernel: `find_top_clusters_kernel`

end for

Phase 2: 查询分组与重排

根据访问簇将查询分组：

$$g_{c_j} = \{\, q_i \mid c_j \in clusters_i \},\quad j = 1, \ldots, nlist$$

Phase 3: 分组矩阵运算

for each 簇组 (c_j, g_{c_j}) **in parallel do**

if $|V_{c_j}| > 0$ **and** $|g_{c_j}| > 0$ **then**

构建查询子矩阵 $Q_{\text{sub}} = [\, q_i \mid q_i \in g_{c_j} \,]$

执行批量矩阵乘法：

$$S = Q_{\text{sub}} \times V_{c_j}^{\top}$$

▷ 使用 cuBLAS 优化的 SGEMM 操作

转换内积为距离：

$$D_{\text{dist}} = 1 - S$$

GPU 并行 Top-K 选择：

$$R_{\text{sub}} = \text{GPU-TopK}(D_{\text{dist}},\, k)$$

▷ kernel: `gpu_topk_for_cluster`

将 R_{sub} 映射回原查询索引并合并到 R

end if

end for

Phase 4: 结果聚合

对每个查询的多簇结果执行最终 Top-K 合并 **return** R

2. 不同分组方式尝试

为了最大化查询间并行的效果，我们尝试实现并评估了多种查询分组策略，希望能将进一步提升运行效率[1]

1. **基准策略：无分组**简单顺序分批，将查询等分为固定大小的批次，作为性能基准。
2. **基于簇相似性的分组**先计算每个查询的 $nprobe$ 个最近簇，以主要簇（最近簇）为键，将访问相同主要簇的查询分到同一批次。

[1]篇幅所限无法展开具体的算法实现，详情可以查看上述代码仓库

3. **基于查询向量相似性的分组**对查询向量使用 K-means（5 轮）聚类，将相似向量聚到同一批次，利用内积距离度量相似性。
4. **混合分组策略**先按簇相似性粗分，再在每个粗组内按查询相似性细分，实现内存访问和计算局部性的双重优化。
5. **自适应批大小策略**根据查询与邻近查询的相似度动态调节批大小：高相似性增大批量以提吞吐，低相似性缩小批量以减冲突。
6. **负载均衡分组**按查询访问簇数估算复杂度，贪心分配查询到当前最轻批次，避免计算热点。
7. **局部性感知分组**在邻域范围内搜索相似查询，限制搜索半径以降低分组开销，同时提升缓存命中。
8. **分层分组策略**第一级按主要簇分，第二级在组内按查询相似度细分，结合簇访问和向量相似性。
9. **时间感知自适应分组**基于查询向量的统计特征（方差等）预测其计算复杂度，自适应分配批大小。

主要评估指标：latency、簇重叠率、recall。

四、 实验设计

(一) 数据集处理与评估指标

本实验使用 DEEP100K 数据集，实现并测试不同 MPI 并行优化方法在 ANN 中的性能表现。以下是详细的实验设计与实现：

1. **数据集分割**
   - 使用 DEEP100K 数据集（100,000 个 96 维向量）。
   - 从 query 文件中选取前 X 条作为查询集；base 文件作为基础数据集。
   - 使用预先计算好的精确 K 近邻（GT）数据作为基准。
2. **评估指标**
   - **主要指标**：查询延迟（latency，毫秒）。
   - **约束条件**：召回率（recall@10）≥ 0.9。

(二) 调用测试框架进行测试

注意由于硬件需要暖机，我们在正式统计前预先进行 100 条查询暖机；正式测试时，采用 5 次重复取平均值的方法以消除偶发抖动以保证测试公平性。

五、 实验结果分析

(一) GPU 批次加速方案加速比

我们把不同的加速比的实验分析结果展示如下图1

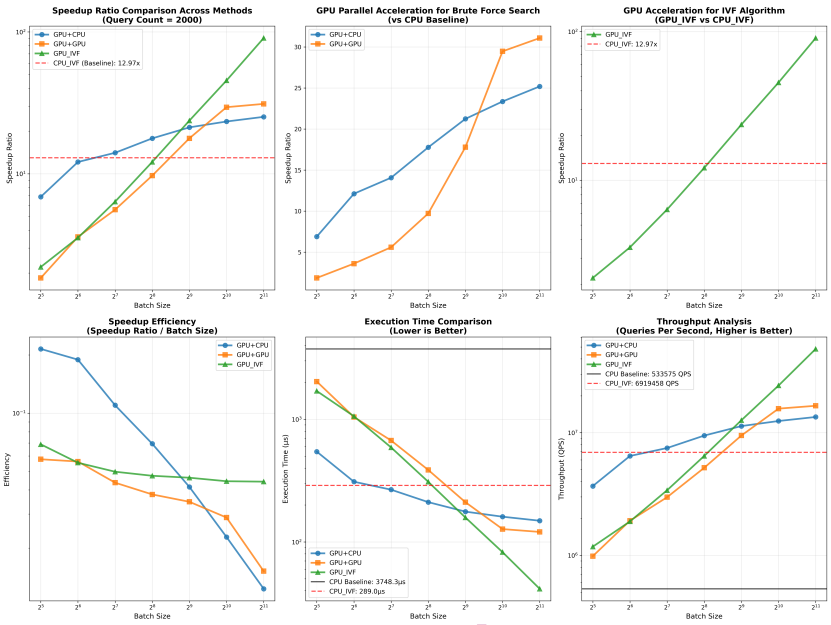

图 1: GPU 批次加速对比实验结果

1. 相比 CPU 实现 GPU 的加速效应

从可视化结果可以观察到:

- **稳定性对比**: GPU+CPU 表现出更好的线性增长特性，而 GPU+GPU 呈现指数型性能增长，后续我们将深入分析这个问题
- **峰值性能**: GPU+GPU 在大批量处理时达到 31.08 倍的峰值加速比，超越 GPU+CPU 的 25.19 倍

2. GPU 批次对 IVF 算法的加速分析

GPU_IVF 算法结合了 IVF 索引的智能性和 GPU 并行计算的强大能力，实现了所有测试方案中的最优性能:

- **性能交叉点**: GPU_IVF 在批大小 256 附近 (12.12 倍) 开始接近 CPU_IVF 的性能 (12.97 倍)
- **超越阈值**: 从批大小 512 开始，GPU_IVF 显著超越 CPU_IVF，最终实现 6.97 倍的相对加速
- **极致性能**: 在批大小约为 2K 时，GPU_IVF 达到 90.41 倍的极致加速比，相对于 CPU 基准提升 98.9%
- **吞吐量优势**: 峰值吞吐量达到 48,270,677 QPS，是 CPU_IVF 的 7 倍，CPU 基准的 90 倍

3. 讨论：batchsize 与 query 数量分析 GPU 的吞吐优势

实验结果清晰展示了批大小作为 GPU 性能关键因子的重要作用:

- **效率递减规律**: 随着批大小增加，单位批量的加速效率呈递减趋势

- **大批量优势**：从图中可以很明显的发现，GPU 算法基本都在吞吐量较大时开始“发力”，实现效率的提升，这很明显反映了 GPU 并行计算的特性。

(二) CPU 选择 TOP-K 与 GPU 实现 TOP-K 加速对比

我们对比了两种方案的性能：(1) GPU 计算矩阵乘法后使用 CPU 进行 TOP-K 选择（简称 GPU+CPU 方案）和 (2) 完全在 GPU 上进行矩阵运算和 TOP-K 选择（简称 GPU+GPU 方案)。实验通过不同批大小（Batch Size）和查询数量（Query Count）的组合进行了全面测试，以分析两种方案在各种工作负载下的表现，其表现大致如下图2所示

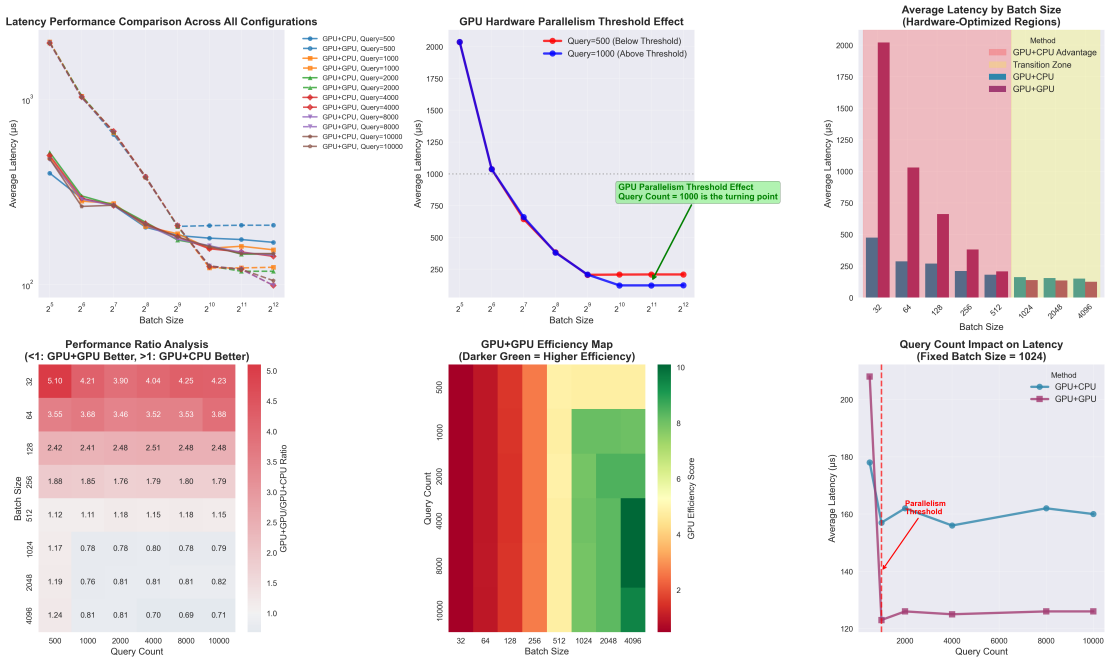

图 2: 不同 TOP-K 选择策略效果对比

1. 召回率分析

从实验数据来看，两种方案在召回准确性方面不存在准确性问题，这表明 GPU 进行 TOP-K 选择不会影响准确搜索的结果质量。

2. 延迟分析

从实验结果可以比较明显的观察到，由于 GPU 在批次较小时由于通信开销明显大于计算开销故 GPU+GPU 方案表现远远不如 GPU+CPU，而后续的实现中我们发现 GPU+CPU 更早触及边际效应，GPU+GPU 方案方案表现则快速优化直至比 GPU+CPU 良好，从上述折线图可以非常明显地看到这个现象。

3. 加速比分析

我们可以通过计算两种方案的加速比（GPU+GPU/GPU+CPU）来量化性能差距：

- 在小批量处理时（Batch Size $\leq$ 256），GPU+CPU 方案具有显著优势，延迟比通常大于 3。
- 随着批大小增加到 512，延迟比降至 1.2 左右。

- 当批大小达到 1024 及以上时，延迟比约为 0.7-0.8，表明 GPU+GPU 方案开始展现性能优势。

综上，我们的实验对比了在向量检索任务中，两种不同的 TOP-K 选择方案的实验效果，而我们发现再不同的批次处理条件下对 TOP-K 的选择情况，这与我们在第二次实验中探讨的混合并行的思想不谋而合，在后续的工作中我们也将探讨混合并行的可能性。

(三) 不同分组方式对比

我们对 9 种不同的查询分组策略进行全面对比分析。实验涵盖了批大小从 64 到 1024 的范围，通过簇重叠率与延迟的关系揭示各策略的性能特征和适用场景实验结果可视化如下图3所示

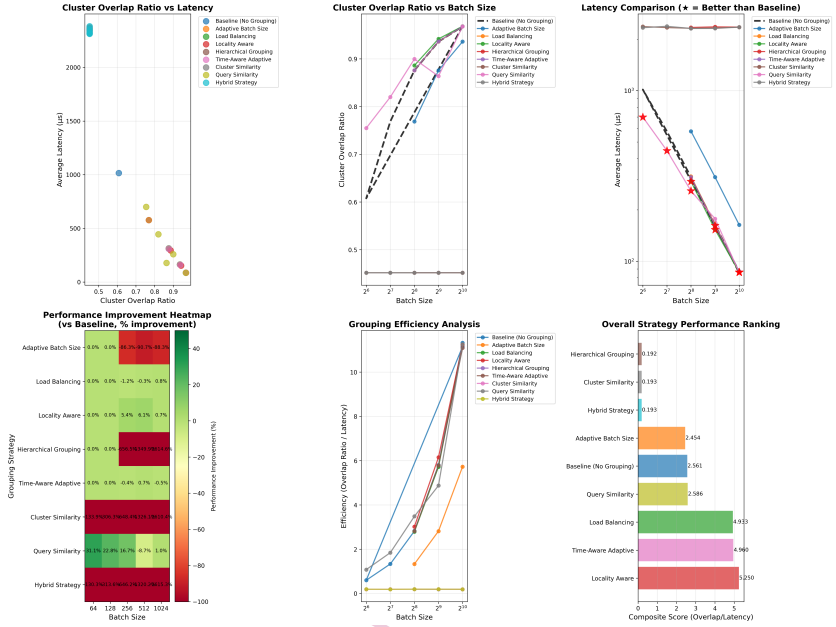

图 3: 不同分组模式对比，其中基线表现特殊标出，以区分不同分组策略的优劣

1. 簇重叠率与延迟关系分析

簇重叠率是衡量分组策略效果的关键指标，它反映了同一批次内查询访问相同簇的程度。理论上，较高的簇重叠率意味着更好的内存访问局部性和缓存利用率，从而带来更低的延迟。

- **高重叠率区间**（>0.9）：局部性感知、时间感知自适应、负载均衡等策略表现优异，延迟控制在 86-294 s 范围内
- **中重叠率区间**（0.6-0.9）：查询相似性分组策略在此区间表现稳定，延迟介于 258-699 s
- **低重叠率区间**（ 0.45）：簇相似性、混合策略和分层分组策略重叠率较低，延迟显著增加至 2300 s 以上

2. 批大小对分组策略的影响

batchsize 会影响分组策略性能，不同策略在不同批大小下的表现差异显著：

- **基准策略**：表现出典型的单调增长模式，重叠率从 0.608（批大小 64）增长至 0.968（批大小 1024），批次增大会导致重叠增大，这是自然且符合直觉的。

- **局部性感知策略**：在所有批大小下均优于基准，重叠率最高达 0.968，大批大小时收敛于基准水平
- **查询相似性策略**：在中等批大小（256-512）时达到最优重叠率 0.900，但在大批大小时收敛于基准水平
- **问题策略**：簇相似性、混合策略和分层分组在所有批大小下均维持较低的重叠率（ 0.452）

3. 策略性能提升分析

通过性能提升热力图分析，可以量化各策略相对于基准的改进程度。实验结果显示，不同策略在各批大小下的表现存在显著差异，其表现与簇重叠率的高低直接相关：

优秀表现策略性能表：

策略	批大小	性能提升 (%)	簇重叠率	适用场景
查询相似性	64	31.1	0.755	小批量优势
	128	22.8	0.820	中小批量
	256	16.7	0.900	中等批量
局部性感知	256	5.4	0.886	中等批量
	512	6.1	0.942	大批量优势
	1024	0.7	0.968	大批量稳定
负载均衡	1024	0.8	0.968	大批量均衡
时间感知自适应	512	0.7	0.937	中大批量

表 1: 优秀策略性能提升详细数据

查询相似性策略在小到中等批大小时表现卓越，最高可获得 31.1% 的性能提升；局部性感知策略在中到大批大小时表现稳定优异。相反，簇相似性、混合策略和分层分组策略由于过度细分导致 GPU 并行度不足，性能表现不佳甚至不如不进行特殊处理的组。

六、 总结

本次实验我们在在统一的 GPU+CUDA 实验环境下，探索了不同维度利用 GPU 对 ANNS 进行加速的方案及优化策略。实验结果表明：

- **批次矩阵化并行**显著提升单机吞吐，峰值加速比可达 >30×，并且 GPU_IVF 在大批量时实现最优 >90×。
- **混合 vs. 纯 GPU TOP-K**：GPU+CPU 方案在延迟敏感、小批量场景更稳健，纯 GPU 方案则在批量足够大时性能超越。
- **分组策略效果**与簇重叠率高度相关：局部性感知与查询相似性聚类在中大批量场景中表现最佳，自适应与负载均衡策略兼顾了吞吐与均衡；过度细分（如分层/混合过度）反而因吞并程度不足导致性能下降。

本次实验我们合理利用 GPU 高吞吐率的特点，尝试在将查询序列分为若干 batch 来进行查询间的并行，显著提升了算法效率，在后续的期末报告中能够更深入的对 ANN 算法进行并行优化[2]。

[2] **AI 的使用情况：**本次实验中，实验数据可视化使用了 AI 工具进行辅助，一些分组方向的提出参考了 ai 的建议