CUVS源码

RAPIDS生态系统中的向量数据库GPU支持
cuvs目前不支持多GPU构建K-means索引

RAFT库

RAFT (RAPIDS Analytics Foundation Toolkit) **NVIDIA RAPIDS生态系统中的一个核心C++库**。它是专门为GPU加速的数据科学和机器学习工作负载设计的。

RAFT在cuvs中的主要用途：

1. **内存管理**:
   - `raft::device_mdarray` - GPU设备内存数组
   - `raft::host_mdarray` - 主机内存数组
   - `raft::device_vector` - GPU设备向量
2. **线性代数操作**:
   - `raft::linalg::gemm` - 矩阵乘法
   - `raft::linalg::norm` - 范数计算
   - `raft::linalg::add` - 向量加法
3. **CUDA资源管理**:
   - `raft::resources` - CUDA资源句柄
   - `raft::resource::get_cuda_stream` - 获取CUDA流
4. **数学工具**:
   - `raft::util::pow2_utils` - 2的幂次工具
   - `raft::util::integer_utils` - 整数工具

向量交错存储

cuvs中的向量采用交错存储，而不是连续存储

原理

传统存储方式 (连续存储):

```
向量1: [v1_0, v1_1, v1_2, v1_3, v1_4, v1_5]
向量2: [v2_0, v2_1, v2_2, v2_3, v2_4, v2_5]
向量3: [v3_0, v3_1, v3_2, v3_3, v3_4, v3_5]
```

交错存储方式 (kIndexGroupSize=32, veclen=2):

```
维度0: [v1_0, v1_1, v2_0, v2_1, v3_0, v3_1, ...]
维度2: [v1_2, v1_3, v2_2, v2_3, v3_2, v3_3, ...]
维度4: [v1_4, v1_5, v2_4, v2_5, v3_4, v3_5, ...]
```

优势:

1. **分支友好**: 如果分支条件是维度索引，那么不会造成wrap divergence
2. **向量化加载**: 一次可以加载多个向量的相同维度
3. **缓存友好**: 提高GPU L1/L2缓存的命中率

利用GPU Warp提高并行度

cuvs中的使用示例：

```
using align_warp = raft::Pow2<raft::WarpSize>;  // WarpSize = 32
const int lane_id = align_warp::mod(threadIdx.x);  // 获取线程在warp中的ID (0-31)
```

在ivfflat搜索

```
// 每个warp处理32个向量
for (uint32_t group_id = align_warp::div(threadIdx.x); group_id < num_groups; group_id += kNuml
  // 32个线程同时计算32个向量的距离
  const uint32_t vec_id = group_id * raft::WarpSize + lane_id;

  // 每个线程处理一个向量的距离计算
  if (valid) {
    // 计算距离...
  }
}
```

Warp级优化的优势

1. **内存合并访问**:
   - 32个线程同时访问连续内存
   - 一次内存事务可以服务整个warp
2. **分支效率**:
   - 如果warp内所有线程走相同分支，效率最高
   - 如果分支不同，会产生warp divergence
3. **共享内存优化**:
   - warp内线程可以高效共享数据
   - 使用 `__shfl` 指令进行warp内数据交换

在向量发生变化时更新聚类中心

源码位置: /root/cuvs/cpp/include/cuvs/neighbors/ivf_flat.hpp:42-52

```
bool adaptive_centers = false;
```

设为false：

- 默认情况下，聚类中心在ivf_flat::build中训练，在ivf_flat::extend中从不修改。
- 结果是在调用几次ivf_flat::extend后，可能需要从头重新训练索引。

设为true：

- 更新**新数据所在聚类**的聚类中心
- 聚类依然容易失真

cuvs中k-means聚类训练策略

训练集比例配置

源码位置: /root/cuvs/cpp/include/cuvs/neighbors/ivf_flat.hpp:40

```
struct index_params : cuvs::neighbors::index_params {
    /** The fraction of data to use during iterative kmeans building. */
    double kmeans_trainset_fraction = 0.5;  // 默认使用50%的数据进行训练
    /** The number of iterations searching for kmeans centers (index building). */
    uint32_t kmeans_n_iters = 20;           // 默认20次迭代
};
```

训练集大小计算

源码位置: /root/cuvs/cpp/src/neighbors/ivf_flat/ivf_flat_build.cuh:418-420

```
// 计算训练集比例和大小
auto trainset_ratio = std::max<size_t>(
    1, n_rows / std::max<size_t>(params.kmeans_trainset_fraction * n_rows, index.n_lists()));
auto n_rows_train = n_rows / trainset_ratio;
```

- 基础训练集大小: `kmeans_trainset_fraction * n_rows` (默认50%)
- 最小训练集大小: `n_lists` (聚类数量)
- 实际训练集大小: `max(基础大小, 最小大小)`
- 采样比例: `n_rows / 实际训练集大小`

训练数据采样

- 使用等间隔采样 (stride sampling)，采样间隔: `trainset_ratio`
- 例如: 如果 `trainset_ratio=2` ，则选择第0, 2, 4, 6...行作为训练数据

在向量发生变化时减少索引碎片

交错存储对齐

- 所有列表按32个向量为一组进行交错对齐
- 使用2的幂次对齐减少内存碎片

内存分配策略

源码位置: `/root/cuvs/cpp/include/cuvs/neighbors/ivf_flat.hpp:100-105`

```
constexpr list_spec(uint32_t dim, bool conservative_memory_allocation)
    : dim(dim),
      align_min(kIndexGroupSize),      // 最小对齐: 32
      align_max(conservative_memory_allocation ? kIndexGroupSize : 1024)  // 最大对齐: 32或1024
{
}
```

内存分配模式:

- **保守模式** (`conservative_memory_allocation = true`): 最小对齐32
- **默认模式** (`conservative_memory_allocation = false`): 最大对齐1024

初始化列表容量

源码位置: `/root/cuvs/cpp/src/neighbors/ivf_list.cuh:52-56`

容量计算逻辑:

1. 如果 `n_rows >= align_max` : 按 `align_max` 对齐
2. 如果 `n_rows < align_max` :
    - 取 `max(n_rows, align_min)` 的2的幂次
    - 但不超过 `align_max`

列表扩展时的碎片管理

源码位置: `/root/cuvs/cpp/src/neighbors/ivf_flat/ivf_flat_build.cuh:290-294`

- 新大小: 实际需要的向量数量
- 旧大小: 按32对齐的旧容量

- 避免频繁的小幅扩展