PR #31897 完整报告

sgl-project/sglang

[CPU] refactor rope kernels

合并时间：2026-07-22 09:12

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31897

执行摘要

本 PR 重构了 CPU RoPE 内核，引入 RopeParams + RotaryMode + rotary_embedding_kernel_impl 统一路径，合并三个入口点（rotary_embedding_cpu，apply_rotary_pos_emb_cpu，multimodal_rotary_embedding_cpu），净减少约 500 行重复代码。主要风险是入口函数移除了对 key 维度的验证，可能引入越界错误；但整体可维护性提升，设计模式值得学习。

功能与动机

PR 描述指出重构目标：

- 将三个分散的 CPU RoPE 内核统一为共享路径；
- 删除约 850 行重复的 2D mRoPE 标量内核；
- 跳过 Q/K 共享相同 *cache* 行时的冗余计算；
- 修复 multimodal GQA 形状检查（key head count 可能与 query 不同）。

这些更改旨在减少代码重复、统一实现、降低维护成本。

实现拆解

1. 引入 RopeParams（rope.cpp）：统一管理 2D/3D/4D 张量的维度、步长和偏移计算，自动根据 query.dim() 选择解析路径。
2. 定义 RotaryMode 和缓存行模板：Interleaved/Neox/NeoxFull 三种模式通过枚举区分；SplitCosSinRow 用于普通 RoPE，MropeCosSinRow 用于 2D mRoPE（支持 T/H/W 分片）。
3. 实现 rotary_embedding_kernel_impl：模板内核根据 RotaryMode 特化调用 RotaryEmbedInternal::apply，向量化路径依赖 vec.h 新增的 load_float_vec 函数。
4. 改造三个入口函数：统一调用上述内核，并修复 multimodal 中 key head count 不等于 query 时的形状断言。
5. 新增 load_float_vec（vec.h）：为 bf16/fp16 提供统一的 float 向量加载接口，避免在各内核中重复书写 if constexpr。
6. 测试清理：test_rope.py 删除一处冗余的 assert_close。

以下展示 RopeParams 参数统一化和 RotaryEmbedInternal::apply 向量化实现的代码，注释说明了设计意图。

```
// RopeParams 统一管理 RoPE 所需的维度与步长信息
// 支持 2D/3D/4D 输入，在构造时根据 query 张量的维度数自动推导
struct RopeParams {
  int64_t rotary_dim{0};
  int64_t head_size{0};
  int64_t batches{1}, seqlen{1}, num_heads{1}, num_heads_kv{1};
  int64_t q_strideB{0}, q_strideS{0}, q_strideH{0};
  int64_t k_strideB{0}, k_strideS{0}, k_strideH{0};

  RopeParams(const at::Tensor& query, const at::Tensor& key,
             int64_t head_size_, int64_t rotary_dim_)
      : rotary_dim(rotary_dim_), head_size(head_size_) {
    switch (query.dim()) {
      case 2:
        seqlen = query.size(0);
        num_heads = query.size(1) / head_size;
        num_heads_kv = key.size(1) / head_size;
        q_strideS = query.stride(0);
        k_strideS = key.stride(0);
        q_strideH = head_size;
        k_strideH = head_size;
        break;
      case 3:
        seqlen = query.size(0);
        num_heads = query.size(1);
        num_heads_kv = key.size(1);
        q_strideS = query.stride(0);
        k_strideS = key.stride(0);
        q_strideH = query.stride(1);
        k_strideH = key.stride(1);
        break;
      case 4:
        batches = query.size(0);
        seqlen = query.size(1);
        num_heads = query.size(2);
        num_heads_kv = key.size(2);
        q_strideB = query.stride(0);
        k_strideB = key.stride(0);
        q_strideS = query.stride(1);
        k_strideS = key.stride(1);
        q_strideH = query.stride(2);
        k_strideH = key.stride(2);
        break;
    }
  }
  // ... 偏移计算函数（q_offset, k_offset, q_out_offset, k_out_offset）
};

// Interleaved 模式的 RoPE 向量化实现
```

```
// 特化 RotaryEmbedInternal 模板，使用 vec.h 中的 load_float_vec2 和 load_float_vec
template <>
struct RotaryEmbedInternal<scalar_t, RotaryMode::Interleaved> {
  static inline void apply(
      scalar_t* __restrict__ out,
      const scalar_t* __restrict__ input,
      const scalar_t* __restrict__ cache,
      int size) {
    constexpr int kVecSize = at::vec::Vectorized<scalar_t>::size();
    const int half_size = size / 2;
    int d = 0;
    // 向量化主循环：一次处理 kVecSize 个元素（即 kVecSize/2 个 pair）
    for (; d <= size - kVecSize; d += kVecSize) {
      // 加载相邻 pair（x, y）并转换为 float
      auto [xy0, xy1] = load_float_vec2(input + d);
      // 解交错得到 x 向量和 y 向量
      auto [x, y] = at::vec::deinterleave2(xy0, xy1);
      // 从缓存加载 cos 和 sin 值
      auto cos = load_float_vec(cache + d / 2);
      auto sin = load_float_vec(cache + half_size + d / 2);
      auto out0 = x * cos - y * sin;
      auto out1 = y * cos + x * sin;
      // 重新交错存回
      std::tie(xy0, xy1) = at::vec::interleave2(out0, out1);
      // 转换回原始精度并存储
      convert_from_float_ext<scalar_t>(xy0, xy1).store(out + d);
    }
    // 标量回退循环处理剩余元素（当 size 不是 kVecSize 的倍数时）
    for (; d < size; d += 2) {
      float x = input[d], y = input[d + 1];
      float cos = cache[d >> 1], sin = cache[half_size + (d >> 1)];
      out[d] = static_cast<scalar_t>(x * cos - y * sin);
      out[d + 1] = static_cast<scalar_t>(y * cos + x * sin);
    }
  }
  // mRoPE 重载：cos/sin 可能来自不同的 T/H/W 缓存行
  // 通过 MropeCosSinRow::ptr_at(j) 获取对应缓存指针
  static inline void apply(
      scalar_t* __restrict__ out,
      const scalar_t* __restrict__ input,
      MropeCosSinRow<scalar_t> cache,
      int size) {
    // 实现略，原理与上类似，但每个 pair 索引 j 可能引用不同的 cache row
  }
};
```

评论区精华

> Copilot 审查指出：rotary_embedding_cpu 不再验证 key 的 token 维度与 query/positions 一致，RopeParams 仅从 query 推导 seqlen，若 key.size(0) 不同可能导致越界读写。该问题在 PR 中未得到回应或修复。

风险与影响

风险

- 校验缺失（rope.cpp）：入口处移除了对 key 维度的显式检查，可能导致越界访问。
- 重构回归：大量代码替换，边缘情况覆盖可能不足。
- 性能不确定性：未提供基准测试量化性能变化。

影响

- 用户：功能正确，性能可能提升。
- 开发者：代码更简洁，易于扩展。
- 团队：设计模式可推广至其他 CPU 算子。

关联脉络

本次重构是 CPU 内核统一化的一部分，目前尚未发现与其他开放 PR 存在直接依赖。未来若有新的 RoPE 变体（如 YaRN），可基于此模板快速添加。