# PR #45103 完整报告

vllm-project/vllm

[ROCm][DSV4][Perf] Fuse inverse-RoPE and cache bf16 wo_a in o-projection

合并时间：2026-06-13 04:57

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45103

## 执行摘要

- 一句话：融合逆 RoPE 与 wo_a 缓存，ROCm DSV4 *decode* 性能提升 9-16%
- 推荐动作：值得精读，该 PR 展示了如何通过 Triton kernel 融合和权重缓存实现显著性能优化，设计思路清晰，测试充分。尤其是将多步小 kernel 合并为单 launch、以及惰性缓存模式的实践，对其他算子优化有参考价值。

## 功能与动机

从 PR body 可知，ROCm 上 DSV4 的 o-projection（rocm_inv_rope_einsum）逆 RoPE 由约 10 个小 PyTorch kernel 组成（clone, index_select, repeat_interleave, neg, stack, cat, cast），且静态 fp8 wo_a 权重每次步骤都需反量化。这些操作在 profile 中显示为最大的拷贝 / 乘法 kernel，成为 *decode* 性能瓶颈。

## 实现拆解

1. 引入 Triton 融合 kernel：在 rocm_aiter_mla_sparse.py 中定义 _inverse_rope_gptj_kernel（@triton.jit），一次 launch 完成逆 RoPE，直接输出 bf16。该 kernel 假设非 neox 布局、rope_dim 为偶数、连续张量。
2. 替换逆 RoPE 调用：将 rocm_inv_rope_einsum 中的 _apply_inv_rope_ref（包含 PyTorch 回退）替换为 _fused_inverse_rope_gptj 调用，并根据 review 建议移除了 _can_use_fused_inv_rope 函数，使融合路径成为唯一路径（通过 assert 确保条件满足）。
3. 缓存反量化 wo_a 权重：在 _get_cached_wo_a_bf16 中，首次调用时将 fp8 权重反量化并存储为模块属性 wo_a._dsv4_wo_a_bf16，后续步骤直接返回缓存，避免每步重复反量化。
4. 添加配套测试：在 test_rocm_triton_attn_dsv4.py 中增加三个测试类：与官方 DeepseekV4ScalingRotaryEmbedding.forward_native 对比的 test_fused_inverse_rope_gptj_matches_rotary_native（参数化 token 数、头数、位置 dtype）、空张量测试 test_fused_inverse_rope_gptj_empty、以及缓存功能测试 test_get_cached_wo_a_bf16_plain_caches。

关键文件：

- vllm/v1/attention/ops/rocm_aiter_mla_sparse.py（模块 注意力层；类别 source；类型 core-logic；符号 _apply_gptj_inv_rope_ref, _inverse_rope_gptj_kernel, _fused_inverse_rope_gptj, _apply_inv_rope_ref）：核心优化实现文件，包含 Triton 融合 kernel、逆 RoPE 替换、wo_a 缓存逻辑。

- tests/kernels/attention/test_rocm_triton_attn_dsv4.py（模块 测试；类别 test；类型 test-coverage；符号 _make_dsv4_rotary, _inv_rope_via_rotary_native, _FakeWoA, init）：新增三个测试覆盖融合 kernel 正确性（与官方嵌入对比）、空张量边界、缓存功能。

关键符号：_inverse_rope_gptj_kernel, _fused_inverse_rope_gptj, _get_cached_wo_a_bf16, rocm_inv_rope_einsum

## 关键源码片段

### vllm/v1/attention/ops/rocm_aiter_mla_sparse.py

核心优化实现文件，包含 Triton 融合 kernel、逆 RoPE 替换、wo_a 缓存逻辑。

```
# vllm/v1/attention/ops/rocm_aiter_mla_sparse.py（片段）

@triton.jit
def _inverse_rope_gptj_kernel(
    o_ptr, # [T, H, D] 输入张量指针
    out_ptr, # [T, H, D] 输出张量指针 (bf16)
    pos_ptr, # [T] 位置索引指针
    cos_sin_ptr, # [P, rope_dim] cos/sin 缓存 (fp32, cos 在前半，sin 在后半)
    s_t, s_h, # 输入行步长（最后一维连续）
    os_t, os_h, # 输出行步长
    cs_stride, # cos_sin_cache 行步长
    NOPE: tl.constexpr, # 非 rope 维度数（直接透传）
    HALF: tl.constexpr, # rope_dim // 2
    BLOCK_NOPE: tl.constexpr,
    BLOCK_HALF: tl.constexpr,
):
    '''融合逆 GPT-J RoPE 在末尾 rope_dim 维度。

    实现与 DeepseekV4ScalingRotaryEmbedding.forward_native(inverse=True)
    相同的计算，但直接以 bf16 写出，替代了约 10 个小 kernel 的链式操作。
    '''
    t = tl.program_id(0)
    h = tl.program_id(1)
    in_base = t * s_t + h * s_h
    out_base = t * os_t + h * os_h

    # NoPE 部分直接透传（转换为 bf16）
    n = tl.arange(0, BLOCK_NOPE)
    nmask = n < NOPE
    vals = tl.load(o_ptr + in_base + n, mask=nmask)
    tl.store(out_ptr + out_base + n, vals.to(tl.bfloat16), mask=nmask)

    # RoPE 部分：out_even = a*cos + b*sin, out_odd = b*cos - a*sin
    # (a = 偶序号输入，b = 奇序号输入；sin 取负以实现逆旋转)
    pos = tl.load(pos_ptr + t).to(tl.int64)
    k = tl.arange(0, BLOCK_HALF)
    kmask = k < HALF
```

```
    a = tl.load(o_ptr + in_base + NOPE + 2 * k, mask=kmask).to(tl.float32)
    b = tl.load(o_ptr + in_base + NOPE + 2 * k + 1, mask=kmask).to(tl.float32)
    cos = tl.load(cos_sin_ptr + pos * cs_stride + k, mask=kmask)
    sin = tl.load(cos_sin_ptr + pos * cs_stride + HALF + k, mask=kmask)
    out_even = a * cos + b * sin
    out_odd = b * cos - a * sin
    tl.store(out_ptr + out_base + NOPE + 2 * k, out_even.to(tl.bfloat16), mask=kmask)
    tl.store(out_ptr + out_base + NOPE + 2 * k + 1, out_odd.to(tl.bfloat16), mask=kmask)


def _get_cached_wo_a_bf16(
    wo_a: torch.nn.Module,
    hidden_dim: int,
    o_lora_rank: int,
    n_local_groups: int,
) -> torch.Tensor:
    '''惰性缓存反量化后的 bf16 wo_a 权重。首次调用时缓存，后续直接返回。'''
    if not hasattr(wo_a, '_dsv4_wo_a_bf16'):
        # 首次：执行反量化并缓存
        if hasattr(wo_a, 'weight_scale_inv'):
            # fp8 权重：乘以 scale 然后转 bf16
            scale = wo_a.weight_scale_inv.view(n_local_groups, o_lora_rank, hidden_dim)
            cached = (wo_a.weight * scale).to(torch.bfloat16)
        else:
            # 非 fp8 权重：直接 reshape 并转 bf16
            cached = wo_a.weight.view(n_local_groups, o_lora_rank, hidden_dim).to(torch.bfloat16)
        wo_a._dsv4_wo_a_bf16 = cached
    return wo_a._dsv4_wo_a_bf16
```

## tests/kernels/attention/test_rocm_triton_attn_dsv4.py

新增三个测试覆盖融合 kernel 正确性（与官方嵌入对比）、空张量边界、缓存功能。

```
# tests/kernels/attention/test_rocm_triton_attn_dsv4.py（片段）

@pytest.mark.parametrize('num_tokens', [1, 7, 64])
@pytest.mark.parametrize('num_heads', [1, 8])
@pytest.mark.parametrize('pos_dtype', [torch.int32, torch.int64])
@torch.inference_mode()
def test_fused_inverse_rope_gptj_matches_rotary_native(
    num_tokens: int, num_heads: int, pos_dtype: torch.dtype, default_vllm_config
) -> None:
    '''验证融合 kernel 与官方旋转嵌入的 forward_native(inverse=True) 一致。

    使用了真实的 DeepseekV4ScalingRotaryEmbedding 实例，确保正确性。
    '''
    from vllm.v1.attention.ops.rocm_aiter_mla_sparse import _fused_inverse_rope_gptj

    device = torch.device('cuda')
    torch.manual_seed(0)
```

```
    # 构造官方旋转嵌入（小规模用于测试）
    rotary_emb = _make_dsv4_rotary(device)
    o = torch.randn(
        num_tokens, num_heads, HEAD_DIM, dtype=torch.bfloat16, device=device
    )
    positions = torch.randint(
        0, _ROTARY_CACHE_LEN, (num_tokens,), dtype=pos_dtype, device=device
    )

    actual = _fused_inverse_rope_gptj(
        o, positions, rotary_emb.cos_sin_cache, ROPE_HEAD_DIM
    )
    expected = _inv_rope_via_rotary_native(rotary_emb, o, positions)

    assert actual.dtype == torch.bfloat16
    assert actual.shape == o.shape
    # NoPE 部分透传，应为精确匹配
    assert torch.equal(actual[..., :NOPE_HEAD_DIM], expected[..., :NOPE_HEAD_DIM])
    # RoPE 部分允许 1~2 ulp 误差（来自 fma 顺序）
    torch.testing.assert_close(actual, expected, atol=2e-2, rtol=2e-2)
```

## 评论区精华

核心 review 讨论集中于：

- tjtanaa提出 _can_use_fused_inv_rope 始终为 True，建议移除回退路径并直接使用融合 kernel。作者响应"helper removed"，删除了相关函数和回退代码。
- dllehr-amd询问测试是否应与官方 DeepseekV4ScalingRotaryEmbedding 对比，作者确认已更新测试，直接使用官方旋转嵌入的 forward_native 作为参考。
- dllehr-amd最终批准。
- 测试与官方旋转嵌入对比 (testing): 作者将测试改为直接实例化官方旋转嵌入并对比 forward_native 输出。
- 移除融合逆 RoPE 的回退路径 (design): 作者删除 _can_use_fused_inv_rope 并将融合路径设为唯一路径，通过 assert 保护。
- 更新测试适配辅助函数移除 (testing): 作者相应更新测试代码。

## 风险与影响

- 风险：风险较低，但需注意：
  - 融合 kernel 假设张量布局连续、rope_dim 偶数、非 neox；若未来模型或配置不符，现有 assert 会导致运行时错误（已无回退）。但 DSV4 始终满足这些条件。
  - 缓存权重到模块属性，若模块被销毁或权重更新，缓存需手动清除（当前设计为惰性缓存，首次调用时填充；若训练 / 权重热更新可能导致状态不一致，但推理场景无影响）。
  - 仅影响 ROCm 上的 DSV4 模型，其他模型无变化。
- 影响：影响范围局限在 ROCm DSV4 模型 *decode* 阶段：

- 性能：并发 4-64 时吞吐量提升 8.9%-15.6%，TPOT 减少 7.9%-13.3%，对延迟敏感场景收益明显。
- 代码：只修改了两个文件，主文件减少约 59 行（删除回退代码），新增约 126 行。
- 可维护性：融合 kernel 和缓存逻辑使算子更高效，但去除了回退路径，对平台兼容性要求更高。
- 风险标记：ROCm 专用优化，无 PyTorch 回退路径，运行时 assert 保护

## 关联脉络

- PR #39457 [V1][Metrics] Add MLA attention metrics for DeepSeek MFU estimation: 同属 DeepSeek 模型在 V1 引擎上的优化，本 PR 关注 o-projection 性能，39457 关注指标测量，共同支撑 DeepSeek 在 ROCm 上的部署和分析。