

PR #48847 完整报告

vllm-project/vllm

[ROCm][CI] Loosen block-FP8 fused MoE test tolerance for large-K shapes

合并时间: 2026-08-07 16:19

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/48847>

执行摘要

- 一句话: 修正 block-FP8 MoE 测试参考, 保留全局紧容差
- 推荐动作: 值得精读。重点看 `torch_w8a8_block_fp8_moe` 的双参考设计 (按 kernel 精度配对参考变体) 以及作者对 #46658 方法的量化论证——这是区分“kernel 误差”与“参考伪影”的教科书级案例。mawong-amd 对 FP8 量化 ULP 差异的分析也有普适参考价值。

功能与动机

PR body 指出 `tests/kernels/moe/test_block_fp8.py::test_w8a8_block_fp8_fused_moe` 在 ROCm/gfx950 上对 large-K ($K=7168$)、large-N ($N \geq 1024$) 的 DeepSeek 风格形状超出基准容差, 最大绝对差约 0.066。作者确认发散是参考伪影而非 kernel 错误: 参考用的 torch 原生 fp8 量化舍入与生产 `per_token_group_quant_fp8` 约有 0.06%-0.10% 元素翻转到相邻 fp8 code, 在 $K=7168$ 上累积; modular kernel 融合 fp32-SiLU 而 `fused_experts` 与参考先舍入 bf16, 约 1-ULP 差异也被放大。mawong-amd 进一步指出 torch 参考用 $A^*(1/s)$ 而 Triton/HIP 用 A/s 的 1-2 ULP 差异。PR 目标是在保持严格基准容差的前提下让参考与 kernel 精度行为位级一致。

实现拆解

1. 定位发散根因: 在 gfx950 上实测确认大 K/N 形状失败是参考伪影而非 kernel 错误, 发散呈宽带小而受限分布 (数千个元素略微超出 0.035 容差, 硬上限约 0.066, 无单点超过 0.08)。
2. 量化路径对齐: `tests/kernels/moe/test_block_fp8.py` 中 `torch_w8a8_block_fp8_moe` 的激活量化从 `native_per_token_group_quant_fp8` 切换到 `per_token_group_quant_fp8` (生产同款内核 op), 并改用 `current_platform.fp8_dtype()` 取代硬编码 `torch.float8_e4m3fn`, 使参考量化与 kernel 位级一致; `matmul` 保持 fp32。
3. SiLU 精度配对: 给参考新增 `silu_fp32` 开关——True 时在 fp32 中计算 SiLU 并直接量化, 匹配 modular Helion kernel 的 `silu_and_mul_per_block_quant`; False 时先舍入 bf16, 匹配 `fused_experts`。测试中对 `m_out` 在 `is_rocm()` and $K \geq 4096$ and $N \geq 1024$ 条件下使用 fp32-SiLU 参考 (`ref_out_m`), 否则复用 `ref_out`。
4. 容差保持不变: 全局统一 `atol=rtol=0.035` ($M \geq 8192$ 时 0.039), 平台分支只切换参考精度, 不做任何按架构放宽。

5. 源码配套守护: vllm/model_executor/layers/fused_moe/experts/aiter_mxfp4_w4a8_moe.py 的 `_get_padding_mask` 在无 forward context 时返回 None (新增 `is_forward_context_available()` 检查), 避免直接 kernel 调用崩溃, 并记录 AITER upstream PR#4530 的清理 TODO。
6. 测试验证: gfx950 上 160 passed、20 skipped、17 warnings, 约 348 秒 (原为 30 failed、130 passed、20 skipped) 。

关键文件:

- tests/kernels/moe/test_block_fp8.py (模块 块级量化; 类别 test; 类型 test-coverage; 符号 torch_w8a8_block_fp8_moe) : 核心变更文件: 重写 torch_w8a8_block_fp8_moe 参考实现, 量化切换到生产 per_token_group_quant_fp8, 新增 silu_fp32 精度配对, 并按平台 / 形状选择参考变体; 容差保持全局统一。
- vllm/model_executor/layers/fused_moe/experts/aiter_mxfp4_w4a8_moe.py (模块 专家层; 类别 source; 类型 data-contract) : 配套源码改动: `_get_padding_mask` 增加无 forward context 守护, 避免直接 kernel 调用崩溃; 同时添加 AITER upstream PR#4530 的清理 TODO。

关键符号: torch_w8a8_block_fp8_moe, _get_padding_mask,
test_w8a8_block_fp8_fused_moe

关键源码片段

tests/kernels/moe/test_block_fp8.py

核心变更文件: 重写 torch_w8a8_block_fp8_moe 参考实现, 量化切换到生产 per_token_group_quant_fp8, 新增 silu_fp32 精度配对, 并按平台 / 形状选择参考变体; 容差保持全局统一。

```
# tests/kernels/moe/test_block_fp8.py (整理后的参考实现与验证分支)
```

```
def torch_w8a8_block_fp8_moe(
    a, w1, w2, w1_s, w2_s, topk_weight, topk_ids, block_shape, silu_fp32=False
):
```

```
    """使用原生 torch 实现的 block-wise fp8 量化 fused MoE 参考。
```

```
    silu_fp32=True 时在 fp32 中计算中间 SiLU 并直接量化, 匹配 modular
    Helion kernel (silu_and_mul_per_block_quant) 的精度行为;
    silu_fp32=False 时先把 SiLU 输出舍入到 bf16, 匹配 fused_experts
    与默认参考。每个 kernel 对照与其精度一致的参考变体验证。
    """
```

```
    B, D = a.shape
    topk = topk_ids.size(1)
    a = a.view(B, -1, D).repeat(1, topk, 1).reshape(-1, D)
    out = torch.zeros(B * topk, w2.shape[1], dtype=a.dtype, device=a.device)
```

```
    topk_weight = topk_weight.view(-1)
    topk_ids = topk_ids.view(-1)
```

```

_, block_k = block_shape[0], block_shape[1]
# 用生产 per_token_group_quant_fp8 kernel 做量化, 参考与 kernel 位级一致;
# 消除约 0.06%-0.10% 元素翻转到相邻 fp8 code 的边界发散 (K=7168 时累积)。
# matmul 仍保持 fp32。
a_q, a_s = per_token_group_quant_fp8(a, block_k, dtype=current_platform.fp8_dtype())
a_q = a_q.to(torch.float32)
for i in range(w1.shape[0]):
    mask = topk_ids == i
    if mask.sum():
        inter_out = native_w8a8_block_matmul(
            a_q[mask], w1[i], a_s[mask], w1_s[i], block_shape, output_dtype=a.dtype
        )
        # silu_fp32=True 时在 fp32 精度下做激活与量化, 与 modular kernel
        # 的 silu_and_mul_per_block_quant 融合路径对齐。
        act_out = SiluAndMul().forward_native(
            inter_out.float() if silu_fp32 else inter_out
        )
        act_out_q, act_out_s = per_token_group_quant_fp8(
            act_out, block_k, dtype=current_platform.fp8_dtype()
        )
        out[mask] = native_w8a8_block_matmul(
            act_out_q, w2[i], act_out_s, w2_s[i], block_shape, output_dtype=a.dtype
        )
return (
    out.view(B, -1, w2.shape[1]) * topk_weight.view(B, -1, 1).to(out.dtype)
).sum(dim=1)

```

验证分支 (整理自 test_w8a8_block_fp8_fused_moe 的断言部分) :

modular 路径在 fp32 中融合 SiLU+ 量化 (silu_and_mul_per_block_quant) ,

而 fused_experts 与默认参考先把 SiLU 舍入 bf16, 大 K/N 下约 1-ULP 差异

会把 m_out 推出基准容差。因此在 ROCm 大 K/N 形状上改用 fp32-SiLU 参考,

容差保持全局统一 (0.035 / 0.039) , 不做按架构放宽。

if current_platform.is_rocm() and K >= 4096 and N >= 1024:

```

    with set_current_vllm_config(vllm_config):
        ref_out_m = torch_w8a8_block_fp8_moe(
            a, w1, w2, quant_config.w1_scale, quant_config.w2_scale,
            topk_weights, topk_ids, block_size, silu_fp32=True
        )

```

else:

```

    ref_out_m = ref_out

```

```

torch.testing.assert_close(out, ref_out, atol=tol, rtol=tol)

```

```

torch.testing.assert_close(m_out, ref_out_m, atol=tol, rtol=tol)

```

[vllm/model_executor/layers/fused_moe/experts/aiter_mxfp4_w4a8_moe.py](#)

配套源码改动: `_get_padding_mask` 增加无 forward context 守护, 避免直接 kernel 调用崩溃; 同时添加 AITER upstream PR#4530 的清理 TODO。

vllm/model_executor/layers/fused_moe/experts/aiter_mxfp4_w4a8_moe.py (整理后)

TODO(akaratz): 在 <https://github.com/ROCm/aiter/pull/4530> 发布并被 vLLM
消费后，删除 _get_padding_mask 与 patch_gating_output 这两个 workaround。

```
def _get_padding_mask() -> torch.Tensor | None:
    """
    获取非 padding(0)/padding(1) 的布尔掩码。

    `slot_mapping < 0` 来自 gpu_model_runner.py 中的
    `slot_mapping[num_tokens_unpadded:num_tokens_padded].fill_(-1)`。

    直接调用 kernel 的路径没有 model runner 的 padding 元数据，
    因此在没有 forward context 时无需掩码，直接返回 None。
    """
    if not is_forward_context_available():
        return None

    forward_context = get_forward_context()

    # model runner v2: 直接使用上下文里的 padding 标记。
    if forward_context.is_padding is not None:
        return forward_context.is_padding

    # model runner v1: 从 slot_mapping 推断 padding 位置。
    slot_mapping = forward_context.slot_mapping

    if isinstance(slot_mapping, list):
        slot_mapping_dict = slot_mapping[0]
    else:
        slot_mapping_dict = slot_mapping

    if isinstance(slot_mapping_dict, dict):
        slot_mapping_sample = next(iter(slot_mapping_dict.values()), None)

    if isinstance(slot_mapping_sample, torch.Tensor):
        return slot_mapping_sample < 0
    else:
        return None
```

评论区精华

1. #46658 方法复用之争：divakar-amd 建议参考 #46658 的数值比对工具（fp8_ulp_distance、outlier-count 等）。作者经实测给出量化反驳：MoE 输出是 bf16 而非 fp8，fp8_ulp_distance 不适用，且 bf16 ULP 距离在近零值上失真（最大约 3 万 ULP 而绝对误差仅约 0.05）；outlier-count 边界因发散呈宽带分布（数千个元素）无法套用；rtol=1e-2 相对容差方案在 14.68M 元素形状上仍有约 587 个元素违反。结论：有界绝对容差 + 修正参考是正确路线。

2. mawong-amd 的数值根因分析：定位两类发散源——其一，torch 参考计算 scale 后以 $A * (1/s)$ 缩放，而 Triton/HIP 直接做 A / s ，`v_rcp_*` 类指令仅约 1 ULP 精度，两者可差 2 ULP；其二，torch 参考与 Triton fused_moe W8A8 block kernel 的中间 dtype 处理顺序不同。建议参考直接改用 Triton/HIP 的 FP8 group quant 函数，作者据此落地为 `per_token_group_quant_fp8`。
 3. fp8 dtype 平台化：AndreasKaratzas 建议用 `current_platform.fp8_dtype()` 取代硬编码 `torch.float8_e4m3fn`，作者采纳并应用到参考的两处量化调用。
- 46658 数值方法是否可复用 (question): 未采用 #46658 的方法，改为有界绝对容差 + 修正参考实现。
 - 参考量化与 kernel 位级一致性 (correctness): 作者落地为 `per_token_group_quant_fp8`，参考量化与 kernel 位级一致，并通过实测试验证。
 - fp8 dtype 的平台化写法 (design): 作者采纳并应用到参考的两处量化调用。

风险与影响

- 风险：
 1. 参考与生产量化共享内核：参考改用 `per_token_group_quant_fp8` 后，若该量化内核本身有 bug，参考会与 kernel 同错同漏，降低测试对量化路径错误的敏感度——这是测试盲区风险。
 2. 平台分支覆盖有限：fp32-SiLU 参考仅在 `is_rocm()` and $K \geq 4096$ and $N \geq 1024$ 生效，若 CUDA 侧 modular kernel 同样存在 SiLU 精度差异，该分支不覆盖（当前 CUDA 测试全绿，风险较低）。
 3. 测试成本上升：大 K/N 形状上额外计算 fp32-SiLU 参考，测试运行时间增加（实测约 348 秒），对 CI 时长有轻微影响。
 4. AITER 行为变更：`_get_padding_mask` 在无 forward context 时返回 None，`patch_gating_output` 在 `is_padding` is None 时跳过 padding 修补——这是保护性改动，但需确认不影响有 forward context 的常规路径（当前实现保持原有行为）。- 影响：对用户：无生产推理行为变化，AITER MXFP4 kernel 的无 context 直接调用场景更稳健。对系统：ROCm/gfx950 CI 稳定性提升，block-FP8 MoE 大 K/N 形状不再因参考不符而误报失败。对团队：确立“修参考而非放宽容差”的测试评审标准，为大 K 内核测试的参考精度配对提供可复用范式，避免平台特化容差扩散。- 风险标记：测试参考与生产量化共享内核形成盲区，fp32-SiLU 分支仅覆盖 ROCm 大 K/N 形状，参考新增精度分支提高维护成本

关联脉络

- PR #46658（评论中引用的类似数值发散测试 PR，标题未在上下文中提供）：review 中 divakar-amd 提及的同类数值发散处理方法，作者经实测评估后未采用，但同属 FP8 kernel 数值验证方法学问题域。