PR #31727 完整报告

sgl-project/sglang

[AMD] Fix DeepSeek-V4 fused-RMS FP8 scale metadata on gfx950

合并时间：2026-08-02 15:06

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31727

执行摘要

- 一句话：修复 DSV4 fused-RMS FP8 scale 错位，gfx950 精度回归消除
- 推荐动作：值得精读。核心看点： 1) 布局契约（logical vs physical stride）在跨算子调用链中如何被破坏与修复； 2) producer 端修复优于 consumer 端修复的设计判断，有 #31617 失败案例佐证； 3) torch.as_strided 零拷贝修复技巧与配套的存储别名断言测试。对 AMD 量化和多拓扑部署团队有直接参考价值。

功能与动机

关联 issue #31490 报告 DeepSeek-V4-Flash-FP8 在 dp8ep8 拓扑下 GSM8K 从约 0.925 波段跌至约 0.912，紧贴 0.91 的 CI 门槛，并通过源码 overlay 二分定位到 #29275 的 commit 8d2b66fd。该 commit 为 gfx95 bpreshuffle GEMM 引入 transpose_scale=False 加显式 materialize 的 scale 布局流程，但在 DeepSeek-V4 独立的 fused-RMS FP8 producer 上未兑现布局契约。PR body 说明在 TP4/DP4/EP4 上的排查已排除 Triton/collective 路径，240 个相关 GEMM 全部走 CK，指向 fused-RMS producer 的 scale 元数据错位。

实现拆解

实现拆解如下：

1. 定位生产端契约违约。AITER 的 fused_rms_fp8_group_quant 在 transpose_scale=True 时把 scale 写成 CK 就绪的转置物理布局，却返回行主序外观的 [M, G] 元数据。对逻辑 scale [[a,b,c],[d,e,f]]，物理存储是 [a,d,b,e,c,f]，需要 stride (1, M) 才能正确阅读。
2. 新增零拷贝修复原语。fp8_utils.py 新增 view_aiter_fused_rms_transposed_fp8_scale，对二维 scale 用 torch.as_strided 把 stride 重解释为 (1, M)，不改物理字节；非二维输入原样返回。它与既有 materialize_bpreshuffle_fp8_scale 形成配对：view 负责生产端元数据纠正，materialize 负责消费端物理布局落地。
3. 在 DeepSeek-V4 生产端注入。deepseek_v4.py 的 _fused_rmsnorm_fp8_quant 在 _use_aiter_bpreshuffle_gfx95 分支对 x_quant[1] 套用修复，保持 (q_input, x_scale) tuple 契约、BF16 旁路输出不变。此 tuple 只注入 DeepSeek-V4 attention 侧 dense GEMM（MLA q/kv/o 投影），Triton 路径、共享 DeepSeekV2/GLM producer、CK materializer 与 communicator 均未改动。

4. 测试配套。CPU 单测 test/registered/unit/layers/test_fp8_bpreshuffle_scale.py 新增两个用例，用 patch 模拟 AITER 输出，验证值、stride、data_ptr 别名与幂等性；真实算子测试 test/registered/quant/test_fused_rms_fp8_group_quant.py 新增 test_transposed_scale_matches_bpreshuffle_layout_contract，在 GPU 上对比 transpose_scale=True/False 的 AITER 输出，断言修复后与行主序 scale 完全一致且共享同一存储指针，覆盖 M=1 与 M=64、K=1024/4096 的批量化形状。

关键文件：

- python/sglang/srt/layers/quantization/fp8_utils.py（模块 量化层；类别 source；类型 core-logic；符号 view_aiter_fused_rms_transposed_fp8_scale）：新增核心修复原语 view_aiter_fused_rms_transposed_fp8_scale，用 torch.as_strided 零拷贝恢复 AITER 转置 scale 的逻辑 [M, G] 索引。
- python/sglang/srt/models/deepseek_v4.py（模块 模型层；类别 source；类型 data-contract；符号 _fused_rmsnorm_fp8_quant）：修复真正落地处：_fused_rmsnorm_fp8_quant 在 gfx95 分支对 x_quant[1] 注入 stride 修复，保持 tuple 数据契约。
- test/registered/unit/layers/test_fp8_bpreshuffle_scale.py（模块 单元测试；类别 test；类型 test-coverage；符号 test_repairs_aiter_scale_before_downstream_layout_handling, test_deepseek_v4_repairs_fused_rms_scale_at_producer）：CPU 单测新增两个用例，验证修复函数的值、stride、data_ptr 别名与幂等性，并 patch 验证 DeepSeek-V4 生产端接入。
- test/registered/quant/test_fused_rms_fp8_group_quant.py（模块 算子测试；类别 test；类型 test-coverage；符号 test_transposed_scale_matches_bpreshuffle_layout_contract）：真实 AITER 算子测试：对比 transpose_scale=True/False 的输出，验证修复后与行主序 scale 完全一致且共享存储指针，是布局契约的最强证据。

关键符号：view_aiter_fused_rms_transposed_fp8_scale, _fused_rmsnorm_fp8_quant, test_repairs_aiter_scale_before_downstream_layout_handling, test_deepseek_v4_repairs_fused_rms_scale_at_producer, test_transposed_scale_matches_bpreshuffle_layout_contract

关键源码片段

python/sglang/srt/layers/quantization/fp8_utils.py

新增核心修复原语 view_aiter_fused_rms_transposed_fp8_scale，用 torch.as_strided 零拷贝恢复 AITER 转置 scale 的逻辑 [M, G] 索引。

```
# fp8_utils.py：零拷贝修复 AITER fused-RMS 转置 scale 的逻辑索引

def view_aiter_fused_rms_transposed_fp8_scale(scale: torch.Tensor) -> torch.Tensor:
    # AITER transpose_scale=True 写入 CK 就绪的转置物理存储（列主序），
    # 但返回的元数据看起来是行主序 [M, G]。例如逻辑 scale 为 [[a,b,c],[d,e,f]]
    # 时，物理字节是 [a,d,b,e,c,f]，需要 stride (1, M) 才能正确阅读。
    if scale.dim() != 2:
        return scale
```

```
    # torch.as_strided 只重解释元数据、不复制字节：
    # 存储指针不变，stride 变为列主序 (1, M)，即恢复逻辑索引。
    return torch.as_strided(scale, scale.shape, (1, scale.shape[0]))
```

python/sglang/srt/models/deepseek_v4.py

修复真正落地处：_fused_rmsnorm_fp8_quant 在 gfx95 分支对 x_quant[1] 注入 stride 修复，保持 tuple 数据契约。

```
# deepseek_v4.py：修复必须在 producer 端完成，再喂给 attention 侧 dense GEMM

def _fused_rmsnorm_fp8_quant(hidden_states, weight, eps):
    x_quant, x_bf16, _, _ = fused_rms_fp8_group_quant(
        hidden_states,
        weight,
        eps,
        inp2=None,
        inp2_weight=None,
        inp2_epsilon=None,
        group_size=128,
        dtype_quant=torch.float8_e4m3fn,
        res1=None,
        output_unquantized_inp1=True,
        transpose_scale=_use_aiter_bpreshuffle_gfx95, # gfx95 下 AITER 写转置存储
    )
    if _use_aiter_bpreshuffle_gfx95:
        # 此 tuple 直接注入 MLA q/kv/o 等 dense GEMM，CK materializer 会信任
        # scale 的 stride 元数据；只在消费端修复（如 #31617 的 contiguous）
        # 无效，因为误关联的元数据在此处就已生成。
        x_quant = (
            x_quant[0],
            view_aiter_fused_rms_transposed_fp8_scale(x_quant[1]),
        )
    return x_quant, x_bf16
```

test/registered/quant/test_fused_rms_fp8_group_quant.py

真实 AITER 算子测试：对比 transpose_scale=True/False 的输出，验证修复后与行主序 scale 完全一致且共享存储指针，是布局契约的最强证据。

```
# test_fused_rms_fp8_group_quant.py：真实 AITER 算子上的布局契约测试

def test_transposed_scale_matches_bpreshuffle_layout_contract(self):
    from aiter.ops.triton.fused_fp8_quant import fused_rms_fp8_group_quant

    common_kwargs = dict(inp2=None, inp2_weight=None, inp2_epsilon=None,
                         group_size=128, dtype_quant=torch.float8_e4m3fn,
                         res1=None, output_unquantized_inp1=False)

    for m, k in ((1, 1024), (64, 1024), (1, 4096), (64, 4096)):
        with self.subTest(m=m, k=k):
```

```
x = torch.randn(m, k, dtype=torch.bfloat16, device="cuda")
weight = torch.ones(k, dtype=torch.float32, device="cuda")

# 同一输入分别以两种模式量化，作为互相参照的契约基准
(q_row_major, scale_row_major), *_ = fused_rms_fp8_group_quant(
    x, weight, 1e-6, transpose_scale=False, **common_kwargs)
(q_transposed, scale_transposed), *_ = fused_rms_fp8_group_quant(
    x, weight, 1e-6, transpose_scale=True, **common_kwargs)

repaired = view_aiter_fused_rms_transposed_fp8_scale(scale_transposed)
materialized = materialize_bpreshuffle_fp8_scale(repaired)

# 断言：修复后值、stride、物理存储指针（零拷贝）全部符合契约
torch.testing.assert_close(q_transposed, q_row_major, rtol=0, atol=0)
torch.testing.assert_close(repaired, scale_row_major, rtol=0, atol=0)
torch.testing.assert_close(materialized, scale_row_major, rtol=0, atol=0)
self.assertEqual(repaired.stride(), (1, repaired.shape[0]))
self.assertEqual(repaired.data_ptr(), scale_transposed.data_ptr())
```

评论区精华

讨论中的核心交锋与结论：

- michaelzhang-ai 解释了此前消费端修复失败的根因：误关联的 scale 元数据起源于上游 _fused_rmsnorm_fp8_quant，而非 aiter_w8a8_block_fp8_linear，因此他此前的 #31617（消费端 x_scale.contiguous()）无法生效；本 PR 改在 producer 端修复才正确。
- PR body 标注精确的 TP8/DP8/EP8 MORI 两节点验证待补，michaelzhang-ai 用 MI355X 2N 1P1D disagg 工作流补测，FP8 0.923、FP4 0.928 双 PASS。
- Lzy17 独立复验覆盖 EP16 2P1D 宽拓扑（4 节点），四个 Flash leg 从约 0.90-0.92 恢复到 0.924-0.933，dsv4pro 无回退。
- XinyuJiangCMU 从 RL 训练场景给出跨负载证据：train_rollout_logprob_abs_diff 从 0.2143 降至 0.0438，与已知良好基线 0.0445 一致。
- amd-bot 汇总 CI：本 PR 相关用例（CPU 单测、MI300 真实算子、gfx950 runner）全部通过，其余失败均为基础设施 / 环境问题。
- 生产端修复 vs 消费端修复的根因判断 (design): 确认生产端修复是正确位置，view_aiter_fused_rms_transposed_fp8_scale 在 fused-RMS 输出处纠正 stride。
- 两节点 TP8/DP8/EP8 验收门 (testing): FP8 0.923、FP4 0.928 双 PASS，回到约 0.925 波段，与 TP4/DP4/EP4 结论一致。
- EP16 2P1D 宽拓扑独立复验 (testing): 四个 Flash leg 从约 0.90-0.92 恢复到 0.924-0.933，dsv4pro 无回退。
- RL 训练场景的跨负载证据 (correctness): step 0 的 abs_diff 从 0.2143 降至 0.0438，与已知良好基线 0.0445 一致。
- CI 失败归属与合并判定 (other): 判定已合并（1685d29f），无归因于本 PR 的 CI 失败。

风险与影响

- 风险：
  1. 覆盖面：修复只作用于 _use_aiter_bpreshuffle_gfx95 分支的 DeepSeek-V4 fused-RMS producer。若 DeepSeekV2/GLM 等其他模型存在同样的 AITER fused-RMS + CK materializer 组合，仍可能有同类隐患；PR 作者声明共享 producer 路径不受影响，但 issue #31490 曾提示 #29275 波及 shared 路径，此处存在不确定。
  2. 零拷贝视图：torch.as_strided 不复制数据，修复后的 scale 与原 AITER 输出共享存储；若下游未来出现 in-place 写 scale 的逻辑，会反写 AITER buffer。当前所有下游只读，风险可控但需在后续改动中留意。
  3. M=1 等价性：M=1 时 stride (1, M) 与行主序相同，数值与行为完全不变，测试已覆盖。
  4. 精度敏感型回归：修复效果以端到端 GSM8K 实测为准，不同批大小 / 拓扑下有统计波动；CI 中精确的 TP8/DP8/EP8 两节点场景未自动化，依赖手工验收记录。 - 影响：影响面集中在 AMD gfx950（MI350X/MI355X）上 DeepSeek-V4-Flash FP8/FP4 的推理与 RL rollout 路径，覆盖 DP-attention、EP16 等拓扑；对 NVIDIA/CPU 与其他模型零影响。性能实测无回退（TPOT 与输出吞吐在噪声范围内）。团队侧收益是让此前在 0.91 门槛上闪烁的两个 dp8ep8 leg 稳定回到约 0.925 波段，消除推理、RL、nightly CI 多团队共同面对的精度隐患。 - 风险标记：精度敏感回归修复，gfx95/AITER 分支专属，零拷贝共享存储，仅覆盖 DeepSeek-V4 producer

关联脉络

- PR #29275 [AMD] Fix gfx95 bpreshuffle FP8 activation scale layout: 引入 scale 布局新契约的源头 commit，本 PR 修复其在 DeepSeek-V4 fused-RMS producer 场景漏掉的契约兑现（issue #31490 根因）。
- PR #31617 [AMD] consumer-side x_scale.contiguous() fix attempt: 讨论中提及的消费端修复尝试（标题为讨论转述），因元数据误关联起源于生产端而无效，反证本 PR 生产端修复的必要性。
- PR #32839 （讨论中提及的关联 CI/ 修复 PR）：hdt98 在评论中请求对 #32839 触发 CI，属于同一修复线的后续验证。