PR #33165 完整报告

sgl-project/sglang

[AMD] DeepSeek-V4 MI355X: eliminate bpreshuffle fp8-scale relayout copy in dense w8a8 linear

合并时间：2026-08-19 18:02

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/33165

执行摘要

- 一句话：DSV4 MI355X dense w8a8 消除 fp8 scale 重排拷贝，bit 级一致
- 推荐动作：值得精读。PR 虽小，但展示了一个典型的「zero-copy 布局优化 + 双层测试防护 + 真实硬件验证」闭环：在热路径中精确理解内核的 scale 字节序，用视图替代拷贝，并通过 bit-exact 对比与 dispatch spy 防止未来改动让覆盖失效。建议 AMD 或内核优化团队重点关注 M == 1 退化分支的处理方式和 CI 门控真实性检查（确保测试真正执行而非 skip）。

功能与动机

在 MI355X 上，CK bpreshuffle w8a8 blockscale GEMM 消费 [num_groups, tokens] 列主序的 per-group activation scale。原实现先按行主序量化，再通过 materialize_bpreshuffle_fp8_scale 重排，导致每个 dense w8a8 GEMM（MLA 的 q/kv/o 投影和 MoE）都产生一次真实拷贝，是 DeepSeek-V4 MI355X decode 耗时剖析中较大的 ELEWISE 开销之一。aiter_per1x128_quant 已支持 transpose_scale=True，可直接产出所需字节序；本 PR 正是要求该能力并配合 stride 元数据修复，去除拷贝，同时将已有的 Triton 行主序视图逻辑收敛到同一共享 helper。

实现拆解

1. 核心逻辑（fp8_utils.py）：在 aiter_w8a8_block_fp8_linear 的 fresh-quant 分支新增 emit_bpreshuffle_scale 门控（materialize_bpreshuffle_scale and input_2d.shape[0] >= 2）。M >= 2 时把 transpose_scale 从 False 改为 emit_bpreshuffle_scale，并将 materialize 替换为 view_aiter_fused_rms_transposed_fp8_scale 的零拷贝 stride 修复；M == 1 保持 materialize，因为 [1, G] 与 [G, 1] 字节序一致，materialize 实际是无操作视图。
2. Triton 分支统一（fp8_utils.py）：把原来内联的 torch.as_strided(x_scale, x_scale.shape, (1, x_scale.shape[0])) 收敛到同一个共享 helper view_aiter_fused_rms_transposed_fp8_scale，让转置 scale 的元数据修复只有一处实现，避免两处逻辑漂移。
3. CPU 契约测试（test_fp8_bpreshuffle_scale.py）：新增 TestBpreshuffleScaleFreshQuantNoCopy，用 _simulate_transpose_scale_emit 模拟 transpose_scale 产出的列主序存储，断言 no-copy 视图与 materialize 结果值一致、stride 为 (1, M)、与生产者 buffer 共享存储不触发分配，并验证 M == 1 走 materialize

时保留自然 (G, 1) stride。

4. MI355X 真实路径测试（test_fp8_bpreshuffle_dense_linear_mi35x.py）：在 stage-b-test-1-gpu-small-amd-mi35x 上执行真实 aiter_per1x128_quant（transpose_scale True vs False）与端到端 aiter_w8a8_block_fp8_linear（新路径 vs 旧路径），断言量化字节、scale 值、stride、data_ptr 共享以及 GEMM 输出 bit 级一致，并通过 mock 断言 dispatch 走 CK gemm_a8w8_blockscale_bpreshuffle 而非 Triton，防止形状列表变化使覆盖静默失效。

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 aiter_w8a8_block_fp8_linear）：核心源码变更。aiter_w8a8_block_fp8_linear 的 fresh-quant 分支新增 emit_bpreshuffle_scale 门控，将 scale 以 transpose_scale=True 直接产出 bpreshuffle 列主序字节序，并用共享 helper 零拷贝修复 stride，消除每次 dense w8a8 GEMM 的 relayout 拷贝；同时将 Triton 分支的行主序视图也统一到同一 helper。
- test/registered/unit/layers/test_fp8_bpreshuffle_dense_linear_mi35x.py（模块 AMD 测试；类别 test；类型 test-coverage；符号 TestDenseBpreshuffleScaleNoCopy, test_quant_producer_scale_equivalence, test_dense_linear_paths_bit_exact）：新增的真实硬件测试。在 MI355X/ROCm 7.2 上验证 aiter_per1x128_quant 的 transpose_scale 产生正确的列主序布局，以及端到端 aiter_w8a8_block_fp8_linear 新旧路径的 GEMM 输出 bit 级一致，并通过 mock 检测 GEMM dispatch 走 CK bpreshuffle 而不是 Triton，确保覆盖不失效。
- test/registered/unit/layers/test_fp8_bpreshuffle_scale.py（模块 量化测试；类别 test；类型 test-coverage；符号 TestBpreshuffleScaleFreshQuantNoCopy, _simulate_transpose_scale_emit, test_nocopy_matches_materialize, test_nocopy_shares_storage_no_allocation）：新增 CPU 契约测试。用 _simulate_transpose_scale_emit 模拟 transpose_scale 产出的列主序存储，pin 住 no-copy 视图与 materialize 路径在值、stride、存储共享上的等价性，以及 M == 1 走 materialize 时的自然 (G, 1) stride，保障核心正确性声明在无 GPU 环境也可见。

关键符号：aiter_w8a8_block_fp8_linear, aiter_per1x128_quant, view_aiter_fused_rms_transposed_fp8_scale, materialize_bpreshuffle_fp8_scale, test_quant_producer_scale_equivalence, test_dense_linear_paths_bit_exact, test_nocopy_matches_materialize, test_nocopy_shares_storage_no_allocation, test_m1_uses_materialize_path_values_and_layout, _simulate_transpose_scale_emit

关键源码片段

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

核心源码变更。aiter_w8a8_block_fp8_linear 的 fresh-quant 分支新增 emit_bpreshuffle_scale 门控，将 scale 以 transpose_scale=True 直接产出 bpreshuffle 列主序字节序，并用共享 helper 零拷贝修复 stride，消除每次 dense w8a8 GEMM 的 relayout 拷贝；同时将 Triton 分支的行主序视图也统一到同一 helper。

```
# fp8_utils.py —— aiter_w8a8_block_fp8_linear 的 fresh-quant 分支
else:
    # 仅当走 CK bpreshuffle GEMM（非 Triton）时才需要 bpreshuffle 布局的 scale。
    materialize_bpreshuffle_scale = _use_aiter_bpreshuffle_gfx95 and not use_triton

    # M >= 2 时，要求量化内核直接以 transpose_scale=True 输出列主序
    # [num_groups, tokens] 字节序，然后用 as_strided 视图修复 strides，
    # 省掉 .t().contiguous().t() 的重排拷贝；M == 1 退回 materialize 路径，
    # 此时 [1, G] 与 [G, 1] 字节序一致，materialize 只是无操作视图。
    emit_bpreshuffle_scale = materialize_bpreshuffle_scale and input_2d.shape[0] >= 2

    q_input, x_scale = aiter_per1x128_quant(
        input_2d,
        quant_dtype=aiter.dtypes.fp8,
        transpose_scale=emit_bpreshuffle_scale,
    )

    if emit_bpreshuffle_scale:
        # 零拷贝：只重解释 strides，不搬移任何字节。
        x_scale = view_aiter_fused_rms_transposed_fp8_scale(x_scale)
    elif materialize_bpreshuffle_scale:
        # 兼容路径：行主序输出 + materialize 重排（M == 1 时为 no-op）。
        x_scale = materialize_bpreshuffle_fp8_scale(x_scale)
```

test/registered/unit/layers/test_fp8_bpreshuffle_dense_linear_mi35x.py

新增的真实硬件测试。在 MI355X/ROCm 7.2 上验证 aiter_per1x128_quant 的 transpose_scale 产生正确的列主序布局，以及端到端 aiter_w8a8_block_fp8_linear 新旧路径的 GEMM 输出 bit 级一致，并通过 mock 检测 GEMM dispatch 走 CK bpreshuffle 而不是 Triton，确保覆盖不失效。

```
# test_fp8_bpreshuffle_dense_linear_mi35x.py —— 真实量化内核层面对比
def test_quant_producer_scale_equivalence(self):
    fp8 = fp8_utils.aiter.dtypes.fp8
    for m in (1, 2, 8, 16):
        with self.subTest(m=m):
            x = self._rand_input(m)

            # 原始路径：行主序发射 scale，再用 materialize 重排（一次拷贝）
            q_f, s_f = fp8_utils.aiter_per1x128_quant(
                x, quant_dtype=fp8, transpose_scale=False
            )
            mat = materialize_bpreshuffle_fp8_scale(s_f)

            # M == 1 时生产代码保留 materialize：单例维 transpose 已连续，
            # materialize 是无操作视图，stride 保持 (G, 1) 且共享存储
            if m < 2:
                self.assertEqual(mat.stride(), (s_f.shape[1], 1))
                self.assertTrue(torch.equal(mat, s_f))
                continue
```

```
self.assertEqual(mat.stride(), (1, m)) # bpreshuffle 列主序

# 优化路径：transpose_scale=True 直接产出列主序，再 zero-copy 修复 stride
q_t, s_t = fp8_utils.aiter_per1x128_quant(
    x, quant_dtype=fp8, transpose_scale=True
)
nocopy = view_aiter_fused_rms_transposed_fp8_scale(s_t)

# 量化字节与布局无关，两条路径必须 bit 级一致；scale 与 materialize
# 等价、stride 为 (1, M)，且 data_ptr 指向生产者 buffer（零分配）
self.assertTrue(
    torch.equal(q_t.view(torch.uint8), q_f.view(torch.uint8)),
    "quantized output differs between transpose_scale paths",
)
self.assertEqual(nocopy.shape, mat.shape)
self.assertTrue(torch.equal(nocopy, mat))
self.assertEqual(nocopy.stride(), (1, m))
self.assertEqual(nocopy.data_ptr(), s_t.data_ptr())
```

评论区精华

主要交锋集中在三层：一是 CPU 测试只能验证 stride 公式、无法捕获真实量化内核布局错误的质疑，最终通过新增 MI355X 真实路径测试解决；二是 M == 1 时 materialize 的 stride 契约被 reviewer 指出并修正（(G, 1) 而非 (1, 1)）；三是 CI 门控要求——先前绿跑在 ROCm 7.0 上两个新测试全部 skip，必须刷新到当前 main 并在 ROCm 7.2 门控上取得真正执行通过的证据。审阅方两次批准均以此为前提，合并者 HaiShaw 以“AITER specific, and CI clean”定调。

- 真实量化 /GEMM 路径覆盖缺失 (testing): 作者新增 test_fp8_bpreshuffle_dense_linear_mi35x.py，在实际 MI355X/ROCm 7.2 上比较新旧 aiter_w8a8_block_fp8_linear 路径，断言 bit 级一致，并用 mock 断言 GEMM 走 CK bpreshuffle 而非 Triton。
- M == 1 时 materialize 的 stride 契约 (correctness): 作者在 torch 2.9.1+rocm7.2 上实测确认，修正测试期望为 (G, 1) 且共享存储，并保留 M == 1 走 materialize 的 fallback。
- CI 门控与真实执行证据 (testing): 作者刷新分支并重跑；1am9trash 确认 base-a-test-cpu 与 stage-b-test-1-gpu-small-amd-mi35x-rocm720 CI 通过。
- 性能收益方向性说明 (performance): 1am9trash 认可：这个改动对每个 GEMM 消除一次 element-wise kernel；reviewer 将 bit 级一致作为主要正确性保证。
- 共享 helper 重构 (design): HaiShaw 合并时评价“AITER specific, and CI clean”。

风险与影响

- 风险：正确性回归风险集中在 python/sglang/srt/layers/quantization/fp8_utils.py：零拷贝依赖 aiter_per1x128_quant(transpose_scale=True) 产出的物理布局与 view_aiter_fused_rms_transposed_fp8_scale 期望严格一致，未来 AITER 版本若调整布局会静默产生错误结果，需依赖 MI355X 真实测试托底。M == 1 退化分支依赖 [1, G] transpose 已连续、materialize 为 no-op 的语义，CPU 测试已将其 pin 住，但对实现细节

变化的敏感性仍然存在。性能数据未确认：TPOT 改善 0.9%–2.0% 为单次、方向性结果，作者明确标注为 preliminary，需重复 A/B 才能作为吞吐声明。环境依赖 gfx95 + aiter + ROCm >= 7.2；不满足条件的系统自动回退旧路径，无收益但无风险。CI 覆盖有效性存在风险：两个新测试依赖 AMD MI35X 门控真实执行，若 CI 跳过或门控配置变化，覆盖会静默失效（此前就出现过 ROCm 7.0 全 skip 的“绿色”结果）。

- 影响：影响范围严格限定为 gfx95（MI355X）上启用 aiter 的 DeepSeek-V4 模型：decode 阶段的 dense w8a8 线性（MLA q/kv/o 投影与 MoE）每个 GEMM 前少一次 scale 重排拷贝，减少一次 HBM 读写和 ELEWISE kernel 启动，TPOT 初步下降 0.9%–2.0%，低并发下收益更明显。NV 路径与无关硬件完全不受影响。对团队的附加价值是确立了「布局级 no-copy + 真实路径 bit-exact + dispatch spy」的测试模式，并把转置 scale 的元数据修复统一到唯一的共享 helper，降低了后续维护的认知负担。
- 风险标记：核心路径变更，AMD gfx95 专用，测试依赖真实硬件，性能收益待确认

关联脉络

- PR #33313 [AMD] DeepSeek-V4: route decode wo_a bf16 batched matmul to aiter batched_gemm_bf16: 同一 AMD/DeepSeek-V4 性能优化线，聚焦 MI355X 上 DSV4 decode 关键路径的 GEMM 路由与算子选择，与本 PR 的 dense w8a8 路径优化相互衔接。