PR #29275 完整报告

sgl-project/sglang

Fix gfx95 bpreshuffle FP8 activation scale layout

合并时间：2026-07-09 01:55

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/29275

执行摘要

- 一句话：修复 gfx950 上 FP8 bpreshuffle GEMM 的 activation scale 布局不匹配
- 推荐动作：该 PR 是 AMD gfx950 平台 block-FP8 推理的关键正确性修复，建议技术管理者优先审阅和合并。对于开发者，值得关注以下设计决策：
- 如何通过 scale.t().contiguous().t() 在保持逻辑值不变的前提下改变物理布局，这是一种常见的 layout materialization 模式。
- 幂等性设计（materialization_is_idempotent_for_bpreshuffle_layout 测试）确保了即使在不同边界多次调用也不出错。
- 与上游 CK 内核的协同事宜（ROCm/rocm-libraries#8639）展示了跨仓库契约的最佳实践。

功能与动机

原始问题记录于 Issue #28685：在 AMD gfx950 上，GLM-5.2-FP8（基于 DeepSeek V3.2/DSA 架构）产生完全错误的输出（GSM8K 准确率 0%）。根因是 block-FP8 线性层路由到 AITER 的 gemm_a8w8_blockscale_bpreshuffle CK 内核时，activation scale 的物理存储布局与 bpreshuffle 内核期望的布局不匹配。虽然逻辑值正确，但每个线性层引入微小误差，累计 78 层后导致输出彻底损坏。此 PR 解决了 SGLang 侧的 scale 布局契约，配合 CK 侧修复（ROCm/rocm-libraries#8639）彻底解决该问题。

实现拆解

本 PR 的实现分为关键步骤：

1. 在核心工具模块添加布局辅助函数：在 python/sglang/srt/layers/quantization/fp8_utils.py 中新增 materialize_bpreshuffle_fp8_scale 和 materialize_bpreshuffle_fp8_scale_tuple。前者对 2D scale 张量执行 scale.t().contiguous().t()，在保持逻辑值不变的前提下将存储布局转换为 bpreshuffle 内核所需的列主序连续布局；后者对 FP8 线性层常用的 (q_input, x_scale, ...) 元组中的 scale 槽位进行转换，保持其余元素不变。
2. 改写通用 block-FP8 线性路由：在 aiter_w8a8_block_fp8_linear 函数中，当 _use_aiter_bpreshuffle_gfx95 且不使用 Triton 回退时，对于已经量化的输入 scale 直接调用 materialize_bpreshuffle_fp8_scale；对于运行时量化路径，不再使用 transpose_scale=True 将布局融合进量化核，而是保持 transpose_scale=False，然后由辅助函数显式 materialize。

3. 在模型前向路径消费者侧应用契约：在 DeepSeekV2/GLM-5.2 共享的若干关键边界——communicator.py（隐藏状态通信）、forward_mla.py（MLA注意力输出到o_proj）、forward_mha.py（MHA q 投影和 kv 投影）以及 deepseek_v2.py（MoE MLP 层）——将所有 transpose_scale=_use_aiter_bpreshuffle_gfx95 替换为 transpose_scale=False，并在量化操作后通过 materialize_bpreshuffle_fp8_scale_tuple 进行布局 materialize。
4. 添加单元测试：新建 test/registered/unit/layers/test_fp8_bpreshuffle_scale.py，包含三个 CPU 回归测试：验证转换后物理存储的 stride 符合预期（materialized.t().is_contiguous()）、幂等性（第二次 materialize 不改变 stride 和数值）、以及 tuple helper 正确处理额外附加的 payload 元组元素（如 BF16 张量）。

关键文件：

- python/sglang/srt/layers/quantization/fp8_utils.py（模块 量化工具；类别 source；类型 core-logic；符号 materialize_bpreshuffle_fp8_scale, materialize_bpreshuffle_fp8_scale_tuple）：核心变更文件，新增了布局 materialize 辅助函数并修改了通用 FP8 线性路由。
- test/registered/unit/layers/test_fp8_bpreshuffle_scale.py（模块 测试；类别 test；类型 test-coverage；符号 TestBpreshuffleScaleMaterialization, test_materializes_transposed_physical_storage, test_materialization_is_idempotent_for_bpreshuffle_layout, test_tuple_helper_keeps_extra_tuple_payload）：新增的单元测试，覆盖辅助函数的关键行为：转换后 stride 正确、幂等性、tuple helper 保持额外 payload。
- python/sglang/srt/models/deepseek_common/attention_forward_methods/forward_mla.py（模块 注意力模块；类别 source；类型 data-contract）：在 MLA 注意力前向路径中应用 scale 布局契约，修改了量化调用点和 import。
- python/sglang/srt/models/deepseek_common/attention_forward_methods/forward_mha.py（模块 注意力模块；类别 source；类型 data-contract）：MHA 注意力前向路径的 scale 布局契约调整，与 forward_mla.py 类似。
- python/sglang/srt/layers/communicator.py（模块 通信器；类别 source；类型 dependency-wiring）：在 FP8 量化的 hidden_states 通信路径中应用布局契约。
- python/sglang/srt/models/deepseek_v2.py（模块 模型层；类别 source；类型 data-contract）：在 MoE MLP 的前向中为 shared expert 的量化 scale 应用布局转换。

关键符号：materialize_bpreshuffle_fp8_scale, materialize_bpreshuffle_fp8_scale_tuple, aiter_w8a8_block_fp8_linear

关键源码片段

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

核心变更文件，新增了布局 materialize 辅助函数并修改了通用 FP8 线性路由。

```
def materialize_bpreshuffle_fp8_scale(scale: torch.Tensor) -> torch.Tensor:
    """根据 gfx95 bpreshuffle GEMM 的要求，materialize 物理 scale 布局。
    通过 .t().contiguous().t() 转换为列主序连续存储，同时保持逻辑值不变。
    """
```

```
    # 只有 2D 张量才需要转换，1D per-tensor 或 per-token scale 直接返回
    return scale.t().contiguous().t() if scale.dim() == 2 else scale


def materialize_bpreshuffle_fp8_scale_tuple(
    value: Tuple[torch.Tensor, ...],
) -> Tuple[torch.Tensor, ...]:
    """对 FP8 (q_input, x_scale, ...) 元组中的 scale 槽位进行布局 materialize。
    保留元组中其他元素（如 bf16 侧输出）不变，避免不必要的拷贝或类型转换。
    """
    return (
        value[0],
        materialize_bpreshuffle_fp8_scale(value[1]),
        *value[2:],
    )


# 在 aiter_w8a8_block_fp8_linear 中，bpreshuffle 路径的 scale 处理关键分支：
if input_scale is not None:
    # 生产者已提供量化输入和 scale
    q_input = input_2d
    x_scale = input_scale
    # gfx95 bpreshuffle 消费者需要显式布局转换
    if _use_aiter_bpreshuffle_gfx95 and not use_triton:
        x_scale = materialize_bpreshuffle_fp8_scale(x_scale)
    # Triton 回退路径只需要调整 stride（零拷贝）
    elif use_triton and _use_aiter_bpreshuffle_gfx95:
        x_scale = torch.as_strided(x_scale, x_scale.shape, (1, x_scale.shape[0]))
else:
    # 运行时量化，不再使用 fused transpose_scale
    q_input, x_scale = aiter_per1x128_quant(
        input_2d,
        quant_dtype=aiter.dtypes.fp8,
        transpose_scale=False, # transpose_scale 设为 False，由 materialize 函数接管
    )
    # 仅 bpreshuffle 路径需要 materialize
    if _use_aiter_bpreshuffle_gfx95 and not use_triton:
        x_scale = materialize_bpreshuffle_fp8_scale(x_scale)
```

test/registered/unit/layers/test_fp8_bpreshuffle_scale.py

新增的单元测试，覆盖辅助函数的关键行为：转换后 stride 正确、幂等性、tuple helper 保持额外 payload。

```
import unittest

import torch

from sglang.srt.layers.quantization.fp8_utils import (
    materialize_bpreshuffle_fp8_scale,
```

```
    materialize_bpreshuffle_fp8_scale_tuple,
)
from sglang.test.ci.ci_register import register_cpu_ci
from sglang.test.test_utils import CustomTestCase

register_cpu_ci(est_time=5, suite="base-a-test-cpu")


class TestBpreshuffleScaleMaterialization(CustomTestCase):
    def test_materializes_transposed_physical_storage(self):
        # 创建一个 3x4 的 scale 张量作为输入
        scale = torch.arange(12, dtype=torch.float32).reshape(3, 4)
        materialized = materialize_bpreshuffle_fp8_scale(scale)
        # 逻辑值必须保持不变
        self.assertTrue(torch.equal(materialized, scale))
        # 形状不变
        self.assertEqual(materialized.shape, scale.shape)
        # 转换后 stride 变为 (1, M)，即转置后是行主序连续的
        self.assertEqual(materialized.stride(), (1, scale.shape[0]))
        # 验证 .t() 后是连续的（列主序存储）
        self.assertTrue(materialized.t().is_contiguous())

    def test_materialization_is_idempotent_for_bpreshuffle_layout(self):
        scale = torch.arange(12, dtype=torch.float32).reshape(3, 4)
        materialized = materialize_bpreshuffle_fp8_scale(scale)
        # 第二次 materialize 应该保持 stride 和数值不变
        rematerialized = materialize_bpreshuffle_fp8_scale(materialized)
        self.assertTrue(torch.equal(rematerialized, scale))
        self.assertEqual(rematerialized.stride(), materialized.stride())

    def test_tuple_helper_keeps_extra_tuple_payload(self):
        q_input = torch.ones((3, 8), dtype=torch.float32)
        scale = torch.arange(12, dtype=torch.float32).reshape(3, 4)
        bf16_side = torch.ones((3, 8), dtype=torch.bfloat16)
        # tuple helper 应只修改 scale 槽位，保留 q_input 和 bf16_side
        q_out, scale_out, bf16_out = materialize_bpreshuffle_fp8_scale_tuple(
            (q_input, scale, bf16_side)
        )
        self.assertIs(q_out, q_input)
        self.assertIs(bf16_out, bf16_side)
        self.assertTrue(torch.equal(scale_out, scale))
        self.assertEqual(scale_out.stride(), (1, scale.shape[0]))


if __name__ == "__main__":
    unittest.main()
```

python/sglang/srt/models/deepseek_common/attention_forward_methods/forward_mla.py

在 MLA 注意力前向路径中应用 scale 布局契约，修改了量化调用点和 import。

```
# 在文件头部新增导入
from sglang.srt.layers.quantization.fp8_utils import (
    materialize_bpreshuffle_fp8_scale_tuple,
)

# 在 forward_absorb_prepare 函数中的修改（仅在 _use_aiter_gfx95 且 q_b_proj 为 FP8 时）：
if self.use_dsa:
    q_quanted, q_lora, k_nope, _ = fused_rms_fp8_group_quant(
        q,
        self.q_a_layernorm.weight,
        self.q_a_layernorm.variance_epsilon,
        k_nope,
        self.kv_a_layernorm.weight,
        self.kv_a_layernorm.variance_epsilon,
        group_size=128,
        dtype_quant=torch.float8_e4m3fn,
        res1=None,
        output_unquantized_inp1=True,
        transpose_scale=False, # 改为 False，不再在量化核内做转置
    )
    # 在 bpreshuffle 路径显式 materialize scale 布局
    if _use_aiter_bpreshuffle_gfx95:
        q_quanted = materialize_bpreshuffle_fp8_scale_tuple(q_quanted)
    q = q_quanted

# 类似修改出现在其他 FP8 量化分支及 o_proj 处理分支（forward_absorb_core 等）
```

评论区精华

Review 中主要讨论了以下关键点：

- contiguous() 的必要性（alexnails 在 fp8_utils.py#L109 提问）：scale.t().contiguous().t() 中的 .contiguous() 是否必需？hdt98 解释如果只有 scale.t().t() 仅仅是视图往返，不会改变底层存储布局；必须插入 .contiguous() 才能实际完成物理布局的 materialize。
- 重复 materialize 的风险（kkHuang-amd 在 fp8_utils.py#L901 提问）：既然多个消费者边界都调用了 materialize，是否会导致 scale 被两次 layout 转换？hdt98 回应已设计为幂等操作：如果 scale 已是预期布局，scale.t() 已是 contiguous，第二次 .contiguous() 是 no-op，且有对应的单元测试覆盖。
- 性能回归考虑（1am9trash 在 PR 整体聊天中提问）：此前 PR #27289 通过 transpose_scale=True 将布局转换融合进量化核，节省约 2-3% decode 性能。本 PR 回退到 transpose_scale=False 再显式 materialize，是否引入额外内核开销？hdt98 提供了速度测试数据，证明显式 materialize 路径在不同模型形状下均未观察到减速，甚至比 bpreshuffle-off 路径更快。

- 外部依赖合并顺序（alexnails、hdt98、HaiShaw 等讨论）：本 PR 与 CK 修复 ROCm/rocm-libraries#8639 之间的关系。hdt98 论证本 PR 代码无硬依赖，可先合；最终团队决定等待 CK 修复落地后再合，以避免用户使用到不完整修复。
- materialize_bpreshuffle_fp8_scale 中 contiguous() 的必要性 (design): 确认 .contiguous() 是必要且正确的。
- materialize 是否会被重复应用导致问题 (correctness): 幂等设计消除了重复调用风险。
- 从 transpose_scale=True 回退为 False 并显式 materialize 的性能影响 (performance): 当前测试显示无性能退化，建议持续监控。
- 本 PR 与外部 CK 修复 PR #8639 的合并顺序 (other): 等待 CK 修复落地后合并。

风险与影响

- 风险：| 风险类别 | 具体内容 | 严重程度 | |----------|----------|----------| | 回归风险 | 改动仅对 _use_aiter_bpreshuffle_gfx95 为 True 的路径（即 gfx950 + ROCm >= 7.2 + AITER 且形状不在 Triton 调优 allowlist 中的路径）生效。其他平台、版本、后端或 Triton 回退路径不受影响。 | 低 | | 性能风险 | 传统 scale.t().contiguous().t() 引入额外数据搬运。但 GLM-5.2 和 DeepSeek-V4 的端到端速度测试均未观察到减速。大规模部署时可进一步监控 TPOT。 | 低 | | 兼容性风险 | 本 PR 的正确性还依赖于 CK 端对 bpreshuffle 内核的修复（ROCm/rocm-libraries#8639）。若用户仅升级 SGLang 而未更新 ROCm/AITER，bpreshuffle 内核本身仍可能产生错误结果。但本 PR 已在 _use_aiter_bpreshuffle_gfx95 门控后，未触及不满足条件的路径。 | 中 | | 测试覆盖风险 | 单元测试仅验证 4x3 的简单形状和幂等性，未覆盖所有可能的 scale 维度（如 1D per-tensor scale）和极端形状。但辅助函数已对 dim != 2 直接返回原 tensor，且生产路径中 scale 均为 2D。 | 低 |
- 影响：用户影响：AMD gfx950（MI350X/MI355X）用户运行 DeepSeek V3.2 架构 block-FP8 模型（如 GLM-5.2-FP8、DeepSeek-V4）时，将获得正确的推理结果。GSM8K 准确率从接近 0 恢复至 0.96。其他硬件平台（NVIDIA、gfx942、XPU）或非 bpreshuffle 路径完全不受影响。

系统影响：bpreshuffle 路径引入一次 scale 张量的显式转换，但根据提供的基准测试，未产生可观察的性能退化。非 bpreshuffle 路径无变化。

团队影响：消除了 gfx950 上 FP8 推理的一个明显的正确性障碍，为 GLM-5.2 在 AMD 上的部署铺平道路。与 CK 团队协同修复展示了跨仓库契约对齐的工作模式。

- 风险标记：核心路径变更，外部依赖未合并（已解决），性能风险监控中

关联脉络

- PR #28471 docs(cookbook): add AMD MI300X/MI325X/MI355X support for GLM-5.2: 该 PR 为 GLM-5.2 添加了 AMD 部署文档和配置，本 PR 修复了该模型在 AMD 上运行时的 FP8 正确性问题。