PR #27289 完整报告

sgl-project/sglang

[ROCm] dsv4: remove the redundant fp8 scale transpose-copy on decode

合并时间：2026-06-09 02:49

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27289

执行摘要

- 一句话：消除 ROCm decode 中冗余的 fp8 scale 转置拷贝，TPOT 降低约 2%
- 推荐动作：本 PR 是一个非常典型的『消除不必要内存拷贝』的性能优化案例，值得学习其中的 torch.as_strided 零拷贝视图技巧，以及通过条件编译保护跨平台兼容性的设计思想。建议所有参与 ROCm 或底层性能优化工作的工程师精读。

功能与动机

在 MI355X DSv4-Pro decode profile 中，每次 fp8 量化后都跟一个 elementwise_kernel_manual_unroll 的拷贝 kernel，每 decode 步有 183 个此类拷贝，占用约 3% 的 GPU 时间。这些拷贝不做量化，只是把 block scale 从 row-major 转置为 bpreshuffle GEMM 所需的 column-major。根因代码中 x_scale = x_scale.transpose(-1, -2).contiguous().view(*x_scale.shape) 导致的显式拷贝。

实现拆解

1. 引入硬件能力标志：在 deepseek_common/utils.py 中公开 _use_aiter_bpreshuffle_gfx95，该标志在 ROCm ≥ 7.2 且 GPU 为 gfx95 时生效。其他文件（forward_mha.py、forward_mla.py、communicator.py、deepseek_v2.py、deepseek_v4.py）导入该标志。
2. 调整量化产出布局：在所有调用 fused_rms_fp8_group_quant 处，新增 transpose_scale=_use_aiter_bpreshuffle_gfx95 参数。当标志为真时，量化 kernel 直接输出 column-major 的 scale 张量。
3. 消除 GEMM 中的冗余转置：在 fp8_utils.py 的 aiter_w8a8_block_fp8_linear 中，当 input_scale 已存在时（预量化），如果当前使用 Triton 且标志为真，则用 torch.as_strided 调整张量的 stride 来模拟 row-major 视图，避免拷贝。当需要即时量化时，直接将 transpose_scale 传给 aiter_per1x128_quant。
4. 调整 import 顺序：通过 isort 调整各文件的导入顺序，满足 lint 要求。
5. 不影响 CUDA 路径：所有改动均被 _use_aiter 和 _use_aiter_gfx95 等条件保护，CUDA 和非 ROCm 路径不会执行新代码。

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 GEMM 函数

aiter_w8a8_block_fp8_linear 的修改，直接消除了多余的 transose-copy，是性能收益的关键实现点。

- python/sglang/srt/models/deepseek_common/attention_forward_methods/forward_mha.py（模块 注意力前向；类别 source；类型 data-contract；符号 fused_rms_fp8_group_quant, _use_aiter_bpreshuffle_gfx95）：在 DSA 和非 DSA 的 fp8 量化调用处传入 transpose_scale 参数，使量化产出布局与 GEMM 一致。
- python/sglang/srt/models/deepseek_v4.py（模块 模型定义；类别 source；类型 data-contract；符号 _fused_rmsnorm_fp8_quant）：导入 _use_aiter_bpreshuffle_gfx95 并在 _fused_rmsnorm_fp8_quant 辅助函数中透传 transpose_scale。
- python/sglang/srt/models/deepseek_common/attention_forward_methods/forward_mla.py（模块 注意力前向；类别 source；类型 data-contract；符号 transpose_scale）：导入 _use_aiter_bpreshuffle_gfx95 并在 MLA 的 fp8 量化调用处传入 transpose_scale。
- python/sglang/srt/models/deepseek_v2.py（模块 模型定义；类别 source；类型 data-contract；符号 transpose_scale）：将 DeepseekV2 模型中某处量化调用的 transpose_scale 参数从 False 改为 _use_aiter_bpreshuffle_gfx95。
- python/sglang/srt/layers/communicator.py（模块 通信层；类别 source；类型 dependency-wiring；符号 prepare_attn, _use_aiter_bpreshuffle_gfx95）：导入 _use_aiter_bpreshuffle_gfx95 并在 prepare_attn 中的两个量化调用处传入 transpose_scale。
- python/sglang/srt/models/deepseek_common/utils.py（模块 公共工具；类别 source；类型 data-contract；符号 _use_aiter_bpreshuffle_gfx95）：定义了 _use_aiter_bpreshuffle_gfx95 常量，是整个功能的条件开关。

关键符号：aiter_w8a8_block_fp8_linear, _fused_rmsnorm_fp8_quant, fused_rms_fp8_group_quant, prepare_attn, forward_normal_prepare, forward_absorb_prepare

关键源码片段

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

核心 GEMM 函数 aiter_w8a8_block_fp8_linear 的修改，直接消除了多余的 transose-copy，是性能收益的关键实现点。

```
def aiter_w8a8_block_fp8_linear(
    input: torch.Tensor,
    weight: torch.Tensor,
    block_size: List[int],
    weight_scale: torch.Tensor,
    input_scale: Optional[torch.Tensor] = None,
    bias: Optional[torch.Tensor] = None,
) -> torch.Tensor:
    # 将输入展平为 2D，记下原始 shape 以便最后恢复
    input_2d = input.view(-1, input.shape[-1])
    output_shape = [*input.shape[:-1], weight.shape[0]]
```

```
n, k = weight.shape

# 根据硬件能力和 GEMM 规模选择实际的 GEMM kernel
# _use_aiter_bpreshuffle_gfx95 ： ROCm >= 7.2 && gfx95，使用 aiter 的 bpreshuffle 库
# _use_aiter_gfx95 ： ROCm < 7.2 的 gfx95，使用 aiter 的 CK/Triton 库
# fallback ： Triton（兼容所有平台）
if _use_aiter_bpreshuffle_gfx95:
    use_triton = use_aiter_triton_gemm_w8a8_tuned_gfx950(n, k)
elif _use_aiter_gfx95:
    use_triton = use_aiter_triton_gemm_w8a8_tuned_gfx950(n, k)
else:
    use_triton = True

if input_scale is not None:
    # 分支 A：input_scale 由上游（如 fused RMSNorm + quant）预量化传入
    q_input = input_2d
    x_scale = input_scale
    # On ROCm >= 7.2, scale is in bpreshuffle's transposed layout.
    # Triton needs a row-major view, so adjust strides only. No copy.
    if use_triton and _use_aiter_bpreshuffle_gfx95:
        # 用 as_strided 改变张量的 stride，不分配新内存
        x_scale = torch.as_strided(
            x_scale, x_scale.shape, (1, x_scale.shape[0])
        )
else:
    # 分支 B：当前步即时量化，调用 aiter 的 per-128 量化 kernel
    q_input, x_scale = aiter_per1x128_quant(
        input_2d,
        quant_dtype=aiter.dtypes.fp8,
        transpose_scale=(_use_aiter_bpreshuffle_gfx95 and not use_triton),
    )

# 选择合适的 GEMM kernel
if use_triton:
    gemm_a8w8_blockscale_op = triton_gemm_a8w8_blockscale
elif _use_aiter_bpreshuffle_gfx95:
    gemm_a8w8_blockscale_op = gemm_a8w8_blockscale_bpreshuffle
else:
    gemm_a8w8_blockscale_op = ck_gemm_a8w8_blockscale

output = gemm_a8w8_blockscale_op(
    q_input, weight, x_scale, weight_scale,
    dtype=torch.bfloat16 if input_scale is not None else input.dtype,
)

if bias is not None:
    output += bias

return output.to(
```

```
        dtype=torch.bfloat16 if input_scale is not None else input_2d.dtype
    ).view(*output_shape)
```

评论区精华

HaiShaw 要求解决冲突后被满足。1am9trash 批准，并指出 CI 失败均为已知问题。关键讨论是 amd-bot 的 CI 状态评论明确指出：PR-CI 中没有实际运行 gfx950 + ROCm ≥ 7.2 的组合，因此本 PR 的变更并未得到真正的功能验证。这是一个值得注意的缺口。

- CI 未覆盖实际硬件路径的风险 (testing): 需要人工确认在生产硬件上的表现；CI 暂无法覆盖该专有硬件配置。
- 冲突解决 (other): 冲突通过合并 main 解决。

风险与影响

- 风险：风险主要在于量化的 scale 布局协议发生了变化：之前所有量化 kernel 产出 row-major scale，现在 bpreshuffle 路径下产出 column-major。任何未同步修改的 scale 消费者（如调试代码、辅助 kernel）可能会出错。本 PR 通过统一在 aiter_w8a8_block_fp8_linear 入口处调整（Triton 路径用 as_strided 反向适配）来规避。但由于 CI 未覆盖实际硬件，该路径在生产中可能存在未发现的回归。此外，transpose_scale 参数在不同 aiter 版本上的行为是否一致也需关注。
- 影响：影响范围仅限于 ROCm ≥ 7.2 + gfx950 的 DSv4-Pro decode 路径。对于该硬件用户，TPOT 改善约 2%，GPU 占用降低约 3%。对其他 GPU（如 CUDA、ROCm < 7.2）完全没有影响。代码修改虽涉及 7 个文件，但每个改动都很小，且全部受条件编译保护。
- 风险标记：CI 未覆盖实际硬件路径，仅 ROCm ≥ 7.2 + gfx950 生效，量表布局协议变更，消费方需同步

关联脉络

- 暂无明显关联 PR