PR #39213 完整报告

vllm-project/vllm

Fix DeepGEMM ep_scatter output address overflow

合并时间：2026-05-06 02:56

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/39213

执行摘要

- 一句话：修复 DeepGEMM EP scatter 输出地址 32-bit 溢出
- 推荐动作：建议合入。该修复精准解决了长序列下 EP scatter 的地址溢出 bug，改动最小且经过本地验证。值得关注的是其设计原则：在保证正确性的前提下，仅扩宽必要的计算路径，避免全局 int64 带来的性能损失。对于使用 DeepGEMM 的 MoE 模型维护者可以精读此 PR。

功能与动机

当 token 数达到 131072、hidden_size 为 6144 时，dest_token_index * output_tensor_stride0 超过 int32 最大值 2,147,483,647，导致输出地址溢出，引发错误。该问题由 issue #39211 报告，独立复现脚本在 --num-tokens 65536 时通过，在 131072 时失败。

实现拆解

1. 定位溢出点：在 _fwd_kernel_ep_scatter_2 中，output_tensor 的行地址计算 dest_token_index * output_tensor_stride0 使用默认的 int32 运算，当乘积超过 2^31-1 时溢出。
2. 扩宽 stride 为 int64：在循环开始前，将 output_tensor_stride0 显式转换为 int64，确保乘法在 64 位空间进行。
3. 扩宽索引为 int64：在每次原子加后，将 dest_token_index 转换为 int64 并命名为 dest_token_index_i64，用于输出地址计算。
4. 保持其他计算 32-bit：输入地址、scale 地址和 output_index 写入仍使用 int32，因为其溢出阈值较高或影响较小。

关键文件：

- vllm/model_executor/layers/fused_moe/deep_gemm_utils.py（模块 MoE 层；类别 source；类型 data-contract；符号 _fwd_kernel_ep_scatter_2）：核心修复文件，修改 Triton kernel 中输出地址计算的数据宽度，将 stride 和索引从 int32 提升至 int64。

关键符号：_fwd_kernel_ep_scatter_2

关键源码片段

vllm/model_executor/layers/fused_moe/deep_gemm_utils.py

核心修复文件，修改 Triton kernel 中输出地址计算的数据宽度，将 stride 和索引从 int32 提升至 int64。

```
# deep_gemm_utils.py — _fwd_kernel_ep_scatter_2 kernel 修复片段

# ... 之前代码不变 ...

    # 将 output_tensor_stride0 提升为 int64，确保后续乘法在 64 位上进行
    output_tensor_stride0 = output_tensor_stride0.to(tl.int64)

    for token_id in range(start_token_id, total_token_num, grid_num):
        to_copy = tl.load(recv_x + token_id * recv_x_stride0 + offset_in, mask=mask)
        to_copy_s = tl.load(
            recv_x_scale + token_id * recv_x_scale_stride0 + offset_in_s, mask=mask_s
        )

        for topk_index in tl.range(0, topk_num, 1, num_stages=4):
            expert_id = tl.load(recv_topk + token_id * recv_topk_stride0 + topk_index)
            if HAS_EXPERT_MAP:
                expert_id = apply_expert_map(expert_id, expert_map)
            if expert_id >= 0:
                dest_token_index = tl.atomic_add(expert_start_loc + expert_id, 1)
                dest_token_index_i64 = dest_token_index.to(tl.int64) # 转为 int64 避免溢出
                tl.store(
                    output_index + token_id * output_index_stride0 + topk_index,
                    dest_token_index, # output_index 使用 int32 写入，足够
                )
                # 使用 int64 版本计算输出行指针
                output_tensor_ptr = (
                    output_tensor + dest_token_index_i64 * output_tensor_stride0
                )
                output_tensor_scale_ptr = (
                    output_tensor_scale + dest_token_index * output_tensor_scale_stride0
                )
                tl.store(output_tensor_ptr + offset_in, to_copy, mask=mask)
                tl.store(output_tensor_scale_ptr + offset_in_s, to_copy_s, mask=mask_s)
```

评论区精华

主要争议点：是否应同时扩宽输入 tensor 和 scale tensor 的地址计算。

- gemini-code-assist[bot]建议将所有相关 stride 都转为 int64，以防其他位置溢出。
- tlrmchlsmth询问 S1ro1 是否应处理输入 tensor。
- S1ro1回应：输入地址溢出需要 hidden_size > 16384 且 token 数 131072 才可能，概率很低，因此保持最小改动。
- 最终决定仅修复输出地址溢出。

性能权衡：tlrmchlsmth 对潜在性能回归表示担忧，但认可改动范围狭窄，可以接受。

- 是否扩宽输入 /scale 地址计算 (design): 仅修复输出地址，其余保持 int32。
- 能否省略 dest_token_index_i64 变量 (design): S1ro1 未直接回复，但最终代码保留了显式转换，可能为了明确意图。

风险与影响

- 风险：回归风险：低。仅修改了 Triton kernel 内四行地址计算，且由同样的 int64 转换模式（Triton 中常见做法）。 未覆盖的溢出：输入和 scale 地址仍使用 int32，在极端配置（hidden_size > 16384 + 131k tokens）下可能溢出，但概率极低。 缺少测试覆盖：没有新增测试用例验证溢出修复，但 issue 中的独立复现脚本已本地验证通过。
- 影响：影响范围：仅影响使用 DeepGEMM 专家并行（EP）且 token 数超过 ~32k 的场景，如长上下文推理。正常规模部署无影响。 性能影响：极小，仅额外增加两次 int32 到 int64 的转换操作，在 Triton kernel 中开销可忽略。
- 风险标记：核心路径变更，缺少测试覆盖

关联脉络

- PR #39211 Report issue: DeepGEMM EP scatter overflow: 关联 issue，描述了实际触发溢出的配置和复现步骤，是此 PR 的直接来源。