PR #45589 完整报告

vllm-project/vllm

[Bugfix] Fix MoE model load OOM in FlashInfer_TRTLLM backend with sleep mode

合并时间：2026-06-17 10:36

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/45589

执行摘要

- 一句话：修复 FlashInfer TRTLLM 后端 sleep 模式下 MoE 权重加载 OOM
- 推荐动作：建议精读该 PR，尤其关注作者如何系统性地定位问题（从 traceback 到 allocator 行为再到临时张量模式分析），以及设计决策（保留 max_split_size_mb=20 而选择优化自身）。对于维护者，建议补充 _copy_permuted_expert_to_block_layout 的单元测试，并关注 flashinfer 后续 API 变更。

功能与动机

Issue #43951 报告了加载 Qwen3-235B-A22B-Instruct-2507 时在 sleep 模式下权重加载阶段 OOM。问题根因是 #41268 引入的 max_split_size_mb=20 分配器策略与 FlashInfer TRTLLM 权重转换模式冲突——后者对 128 个专家和 94 层反复创建 20-50 MiB 的临时张量，而 vLLM 的 CuMem 权重池在 sleep 模式下无法回收临时段，导致 OOM。

实现拆解

1. 在 flashinfer_utils.py 的 convert_moe_weights_to_flashinfer_trtllm_block_layout 中新增内联函数 _copy_permuted_expert_to_block_layout，该函数通过 view 将专家张量重排为块布局，并利用 torch.index_select 的 out 参数直接写入预分配的目标张量。
2. 在函数开头预计算输出张量形状，通过 torch.empty 分配最终的 w13_weights_shuffled_tensor 和 w2_weights_shuffled_tensor。
3. 循环处理每个专家时，不再使用 clone/permute/contiguous/convert_to_block_layout 链式操作和 list/stack，而是直接调用 _copy_permuted_expert_to_block_layout。
4. 将原本在 unquantized.py 中为 TRTLLM 后端单独调用的 swap_w13_to_w31 操作内联到 permute index 计算中（通过 (permute_indices + rows // 2) % rows 实现），避免了额外的一次完整张量交换。
5. 移除对 convert_to_block_layout 的导入，返回预分配张量的 BF16 视图。 配套改动：在 unquantized.py 中删除了 swap_w13_to_w31 的调用行。

关键文件：

- vllm/model_executor/layers/quantization/utils/flashinfer_utils.py（模块 MoE 后端；类别 source；类型 data-contract；符号 _copy_permuted_expert_to_block_layout, convert_moe_weights_to_flashinfer_trtllm_block_layout）：核心变更文件，重写了 MoE 权重转换为 FlashInfer TRTLLM block layout 的函数，引入内联辅助函数并预分配输出张

量，消除中间内存峰值。

- vllm/model_executor/layers/fused_moe/oracle/unquantized.py（模块 未量化 MoE；类别 source；类型 configuration）：删除为 TRTLLM 后端单独调用 swap_w13_to_w31 的行，该操作已内联到转换函数中。

关键符号：_copy_permuted_expert_to_block_layout, convert_moe_weights_to_flashinfer_trtllm_block_layout

关键源码片段

vllm/model_executor/layers/quantization/utils/flashinfer_utils.py

核心变更文件，重写了 MoE 权重转换为 FlashInfer TRTLLM block layout 的函数，引入内联辅助函数并预分配输出张量，消除中间内存峰值。

```
def convert_moe_weights_to_flashinfer_trtllm_block_layout(
    cache_permute_indices: dict[torch.Size, torch.Tensor],
    w13_weight: torch.Tensor,
    w2_weight: torch.Tensor,
) -> tuple[torch.Tensor, torch.Tensor]:
    """Convert expert weights to FlashInfer's block layout.

    预分配输出张量，通过 torch.index_select 直接写入每个专家，
    避免产生中间临时张量，从而解决 sleep 模式下的 OOM 问题。
    """
    # 仅支持 BF16，要求与 flashinfer TRTLLM 内核一致
    if w13_weight.dtype != torch.bfloat16 or w2_weight.dtype != torch.bfloat16:
        raise ValueError("Unquantized Moe Backend FlashInfer TRTLLM requires bfloat16
        weights")
    from flashinfer.fused_moe.core import (
        _maybe_get_cached_w3_w1_permute_indices,
        get_w2_permute_indices_with_cache,
    )
    epilogue_tile_m = 128
    block_k = 128

    num_experts = w13_weight.shape[0]

    def _copy_permuted_expert_to_block_layout(
        out: torch.Tensor,
        expert_uint8: torch.Tensor,
        source_indices: torch.Tensor,
    ) -> None:
        # 将专家张量 reshape 为 (rows, out_rows, block_k) 并转置为 (out_rows, rows, block_k)
        expert_blocks = expert_uint8.view(
            expert_uint8.shape[0], out.shape[0], block_k
        ).permute(1, 0, 2)
        # 使用 index_select 的 out 参数直接写入预分配张量，避免额外拷贝
        torch.index_select(
```

```
        expert_blocks,
        1,
        source_indices.to(expert_uint8.device),
        out=out,
    )

# 计算输出张量形状并预分配
w13_rows, w13_cols = w13_weight[0].view(torch.uint8).shape
w2_rows, w2_cols = w2_weight[0].view(torch.uint8).shape
w13_weights_shuffled_tensor = torch.empty(
    (num_experts, w13_cols // block_k, w13_rows, block_k),
    dtype=torch.uint8,
    device=w13_weight.device,
)
w2_weights_shuffled_tensor = torch.empty(
    (num_experts, w2_cols // block_k, w2_rows, block_k),
    dtype=torch.uint8,
    device=w2_weight.device,
)

for i in range(num_experts):
    w13_expert_uint8 = w13_weight[i].view(torch.uint8)
    permute_indices = _maybe_get_cached_w3_w1_permute_indices(
        cache_permute_indices,
        w13_expert_uint8,
        epilogue_tile_m,
    )
    rows = w13_expert_uint8.shape[0]
    # 内联 swap_w13_to_w31：通过偏移 permute index 实现 W13->W31 的语义交换
    permute_indices = (permute_indices + rows // 2) % rows
    _copy_permuted_expert_to_block_layout(
        w13_weights_shuffled_tensor[i],
        w13_expert_uint8,
        permute_indices,
    )

    w2_expert_uint8 = w2_weight[i].view(torch.uint8)
    permute_indices = get_w2_permute_indices_with_cache(
        cache_permute_indices,
        w2_expert_uint8,
        epilogue_tile_m,
    )
    _copy_permuted_expert_to_block_layout(
        w2_weights_shuffled_tensor[i],
        w2_expert_uint8,
        permute_indices,
    )

return (
```

```
        w13_weights_shuffled_tensor.view(torch.bfloat16),
        w2_weights_shuffled_tensor.view(torch.bfloat16),
    )
```

评论区精华

PR body 详细分析了绕过 max_split_size_mb=20 的几种方案及其缺点，最终决定通过减少临时内存来解决问题。作者明确指出“Changing the global max_split_size_mb value is also not robust”，并坚持保留 #41268 的保护策略。GB200 验证报告（来自 aoshen02）确认该 PR 在真实硬件上解决了 OOM，且输出一致。

- 是否绕过 max_split_size_mb=20 还是优化临时内存 (design): 保持 max_split_size_mb=20 不变，通过重写转换函数消除中间张量来解决 OOM。

风险与影响

- 风险：
  1. 回归风险：新转换路径改变了权重重排方式，虽已验证与旧输出一致，但未覆盖所有模型配置（如不同 TP 大小、hidden_size 组合）。
  2. 测试覆盖缺失：新增的 _copy_permuted_expert_to_block_layout 辅助函数没有对应的单元测试，依赖集成测试覆盖。
  3. 依赖 flashinfer 内部 API：代码调用了 _maybe_get_cached_w3_w1_permute_indices 等非公开函数，flashinfer 版本升级可能破坏兼容性。
  4. 假设固定 block_k=128：若未来 flashinfer 更改 block 大小，需要同步修改。 - 影响：用户：修复了大型 MoE 模型（如 Qwen3-235B、DeepSeek V2/R1）在 sleep 模式下无法加载的阻塞 bug，提升加载成功率。性能：在 dummy 加载测试中，新转换路径速度提升约 4x（从 10.7s 降至 2.4s）。团队：核心 MoE 权重转换逻辑改写，但对外接口不变，不影响其他后端。 - 风险标记：核心路径变更，缺少测试覆盖，依赖 flashinfer 内部 API

关联脉络

- PR #41268 [UX][Bugfix] Fix OOM by setting PyTorch max_split_size_mb during model loading: 该 PR 引入了 max_split_size_mb=20 分配器策略，与本 PR 修复的 OOM 有直接因果关系。