PR #43014 完整报告

vllm-project/vllm

[Perf] Optimize moe permute by pre-allocate buffer, 9~14% kernel performance improvement

合并时间：2026-05-28 21:18

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/43014

执行摘要

- 一句话：通过预分配缓冲区优化 MoE permute，小 batch 提升 9-14%
- 推荐动作：建议值得精读。该 PR 展示了 vLLM 中一个典型的中等复杂度性能优化模式：通过预分配缓冲区减少分配开销。设计上采用了数据类 + 可选参数的渐进式修改，确保向后兼容。C++ 与 Python 协作的缓冲区管理、懒初始化、以及审核发现的 reshape vs view 问题，都具有学习价值。此外，测试中直接断言数据指针相同来验证复用，是一种轻量可靠的验证方式。

功能与动机

在 MoE 层中，permute 操作每次调用都会重新分配多个中间张量，小 batch 下分配开销占比显著。通过预分配并复用这些缓冲区，可以显著减少内存分配与释放的延迟。PR body 中列出的性能数据验证了该优化在小 batch 场景下 9~14% 的 kernel 性能提升，而大 batch 由于计算主导，提升不明显。

实现拆解

1. 数据类设计：在 vllm/model_executor/layers/fused_moe/moe_permute_unpermute.py 中新增 MoEPermuteScratch 数据类，通过 __post_init__ 一次性分配所有需要的中间张量，包括 token_expert_indices、permuted_idx、inv_permuted_idx、expert_first_token_offset、permuted_hidden_states（可选）、排序工作空间等。提供 validate 方法在运行时检查输入是否匹配预分配参数，提供 token_expert_indices_view 和 prepare_topk_ids 方法返回整数张量视图。
2. Python 接口扩展：修改 moe_permute 函数签名，增加可选参数 scratch: MoEPermuteScratch | None。当 scratch 不为 None 时，函数会使用 scratch 中预分配的张量切片，绕过动态分配逻辑。同时新增 moe_permute_unpermute_supported 导出函数供上层判断。
3. C++ 内核增强：在 csrc/moe/torch_bindings.cpp 中注册 moe_permute_with_scratch 和 moe_permute_sort_workspace_size 算子。在 csrc/moe/moe_permute_unpermute_op.cu 中实现 maybe_allocate_tensor 辅助函数，根据是否传入 scratch 缓冲区决定分配或重用，并将排序相关工作空间改为预分配方式，避免每次调用时再创建。
4. 专家层集成：在 vllm/model_executor/layers/fused_moe/experts/cutlass_moe.py 和 fused_humming_moe.py 中为每个 Expert 类增加 _permute_scratch 成员和 _get_permute_scratch() 懒初始化方法。_get_permute_scratch() 在首次调用且内核支持

时构造 MoEPermuteScratch，然后在 apply 或 main_apply 中传递给 moe_permute 或 run_cutlass_moe_fp8 等函数。

5. 测试与基准：新增 test_moe_permute_reuses_scratch_buffers 测试用例，验证连续两次使用相同 scratch 调用 moe_permute 后，输出张量的数据指针相同（即真正复用），且计算结果一致。更新 benchmarks/kernels/benchmark_moe_permute_unpermute.py，在基准循环中复用同个 scratch，以便性能数据反映优化效果。

关键文件：

- vllm/model_executor/layers/fused_moe/moe_permute_unpermute.py（模块 MoE 层；类别 source；类型 data-contract；符号 MoEPermuteScratch, post_init, validate, token_expert_indices_view）：核心变更文件：新增 MoEPermuteScratch 数据类，修改 moe_permute 函数签名以支持可选 scratch。
- vllm/model_executor/layers/fused_moe/experts/cutlass_moe.py（模块 CUTLASS 后端；类别 source；类型 data-contract；符号 _get_permute_scratch）：集成预分配缓冲区到 CUTLASS 专家层，新增 _get_permute_scratch 懒初始化并传递 scratch。
- vllm/model_executor/layers/fused_moe/experts/fused_humming_moe.py（模块 Humming 后端；类别 source；类型 data-contract；符号 _get_permute_scratch）：集成预分配缓冲区到 Humming 专家层，新增 _get_permute_scratch 方法，包含 hidden_size/hidden_dtype 预分配。
- tests/kernels/moe/test_moe_permute_unpermute.py（模块 测试层；类别 test；类型 test-coverage；符号 test_moe_permute_reuses_scratch_buffers）：新增 test_moe_permute_reuses_scratch_buffers 测试，验证 scratch 缓冲区复用正确性。
- csrc/moe/torch_bindings.cpp（模块 C++ 绑定；类别 other；类型 core-logic）：注册新算子 moe_permute_with_scratch 和 moe_permute_sort_workspace_size，扩展 C++ 接口以支持预分配 scratch。

关键符号：MoEPermuteScratch, post_init, validate, token_expert_indices_view, prepare_topk_ids, _get_permute_scratch, maybe_allocate_tensor

关键源码片段

vllm/model_executor/layers/fused_moe/moe_permute_unpermute.py

核心变更文件：新增 MoEPermuteScratch 数据类，修改 moe_permute 函数签名以支持可选 scratch。

```
# SPDX-License-Identifier: Apache-2.0
# SPDX-FileCopyrightText: Copyright contributors to the vLLM project
from dataclasses import dataclass, field
import torch

@dataclass
class MoEPermuteScratch:
    # Reused metadata buffers for repeated grouped-MoE permutes.
    max_num_tokens: int
    topk: int
```

```
num_experts: int
num_local_experts: int
device: torch.device
hidden_size: int | None = None
hidden_dtype: torch.dtype | None = None
token_expert_indices: torch.Tensor = field(init=False)
expert_first_token_offset: torch.Tensor = field(init=False)
permuted_idx: torch.Tensor = field(init=False)
inv_permuted_idx: torch.Tensor = field(init=False)
permuted_hidden_states: torch.Tensor | None = field(init=False, default=None)
sort_workspace: torch.Tensor = field(init=False)
permuted_experts_id: torch.Tensor = field(init=False)
sorted_row_idx: torch.Tensor = field(init=False)
topk_ids_int32: torch.Tensor = field(init=False)
topk_ids_for_sort: torch.Tensor = field(init=False)
max_expanded_rows: int = field(init=False)

def __post_init__(self) -> None:
    # 在构造时一次性分配所有缓冲区，避免重复分配
    self.max_expanded_rows = self.max_num_tokens * self.topk
    self.token_expert_indices = torch.arange(
        self.max_expanded_rows, dtype=torch.int32, device=self.device)
    self.expert_first_token_offset = torch.empty(
        self.num_local_experts + 1, dtype=torch.int64, device=self.device)
    self.permuted_idx = torch.empty(
        self.max_expanded_rows, dtype=torch.int32, device=self.device)
    self.inv_permuted_idx = torch.empty(
        self.max_expanded_rows, dtype=torch.int32, device=self.device)
    if self.hidden_size is not None:
        hidden_numel = self.max_expanded_rows * self.hidden_size
        self.permuted_hidden_states = torch.empty(
            hidden_numel, dtype=self.hidden_dtype, device=self.device)
    self.permuted_experts_id = torch.empty(
        self.max_expanded_rows, dtype=torch.int32, device=self.device)
    self.sorted_row_idx = torch.empty(
        self.max_expanded_rows, dtype=torch.int32, device=self.device)
    self.topk_ids_int32 = torch.empty(
        self.max_expanded_rows, dtype=torch.int32, device=self.device)
    self.topk_ids_for_sort = torch.empty(
        self.max_expanded_rows, dtype=torch.int32, device=self.device)
    # 仅在平台支持时获取排序工作空间大小
    if moe_permute_unpermute_supported():
        sorter_size = torch.ops._moe_C.moe_permute_sort_workspace_size(
            self.max_expanded_rows, self.num_experts)
        self.sort_workspace = torch.empty(
            sorter_size, dtype=torch.int8, device=self.device)

def validate(self, hidden_states: torch.Tensor, topk_ids: torch.Tensor) -> None:
    # 运行时验证张量与预分配配置一致
```

```
        n_token, n_hidden = hidden_states.shape
        assert hidden_states.device == self.device
        assert topk_ids.device == self.device
        assert n_token <= self.max_num_tokens
        assert topk_ids.size(1) == self.topk
        assert topk_ids.size(0) == n_token
        if self.hidden_size is not None:
            assert n_hidden == self.hidden_size
            assert hidden_states.dtype == self.hidden_dtype

    def token_expert_indices_view(self, n_token: int) -> torch.Tensor:
        # 返回当前行数的 token_expert_indices 视图
        return self.token_expert_indices[:n_token * self.topk].view(n_token, self.topk)

    def prepare_topk_ids(self, topk_ids: torch.Tensor) -> torch.Tensor:
        # 将 topk_ids 转换为 int32，使用预分配缓冲区避免分配
        if topk_ids.dtype == torch.int32:
            return topk_ids
        numel = topk_ids.numel()
        topk_ids_int32 = self.topk_ids_int32[:numel].view_as(topk_ids)
        topk_ids_int32.copy_(topk_ids)
        return topk_ids_int32
```

vllm/model_executor/layers/fused_moe/experts/cutlass_moe.py

集成预分配缓冲区到 CUTLASS 专家层，新增 _get_permute_scratch 懒初始化并传递 scratch。

```
# 在 cutlass_moe.py 中，Expert 类新增如下方法：
def _get_permute_scratch(self) -> MoEPermuteScratch | None:
    # 懒初始化 scratch，仅在内核支持时构造以避免在不支持平台上分配
    if self._permute_scratch is None and moe_permute_unpermute_supported():
        self._permute_scratch = MoEPermuteScratch(
            max_num_tokens=self.moe_config.max_num_tokens,
            topk=self.moe_config.experts_per_token,
            num_experts=self.moe_config.num_experts,
            num_local_experts=self.moe_config.num_local_experts,
            device=torch.device(self.moe_config.device),
        )
    return self._permute_scratch

# 并在 apply 方法中传递 scratch：
def apply(self, ...):
    ...
    output = run_cutlass_moe_fp8(
        ...,
        self._get_permute_scratch(),
    )
```

评论区精华

在 Review 过程中，自动代码审查工具 gemini-code-assist[bot] 指出了两个关键问题：

- MoEPermuteScratch.__post_init__ 中直接调用 moe_permute_sort_workspace_size 会在不支持 MoE permute 的平台（如 CUDA < 12.0、ROCm）上导致崩溃。作者后续添加了 moe_permute_unpermute_supported() 守卫，改为条件调用。
- C++ 函数 maybe_allocate_tensor 中使用 reshape 可能返回副本而非视图，导致 scratch 缓冲区的修改不生效。作者将 reshape 替换为 view 并添加了连续性检查。

审核人 bnellnm 进一步询问了若干设计问题：

- permute_scratch 应改为可选参数，以避免所有调用点必须传入。作者修正为 MoEPermuteScratch | None 类型，并在 cutlass_moe.py 和 fused_humming_moe.py 中通过 _get_permute_scratch() 条件提供。
- 建议将部分尺寸断言移至 scratch.validate，作者已采纳。
- 询问了 prepare_topk_ids 中类型转换的测试覆盖，作者认为不需要单独测试。

所有讨论已解决，PR 获得了两名审核人的 Approval。

- moe_permute_sort_workspace_size 在不支持平台上崩溃 (correctness): 作者添加了 moe_permute_unpermute_supported() 守卫，仅在支持时调用该函数。
- C++ reshape 可能返回副本 (correctness): 作者将 reshape 替换为 view，并添加连续性断言，确保返回的是视图。
- permute_scratch 应为可选参数 (design): 作者将类型改为 MoEPermuteScratch | None，并在调用点通过 _get_permute_scratch() 条件返回 None。
- 将尺寸断言移至 scratch.validate (correctness): 作者采纳建议，将尺寸检查整合到 validate 中。

风险与影响

- 风险：
  - 兼容性风险：MoEPermuteScratch 在 CUDA < 12.0 或 ROCm 上初始化时需跳过 moe_permute_sort_workspace_size 调用。当前已在 __post_init__ 中通过 moe_permute_unpermute_supported() 进行守卫，但所有集成点（_get_permute_scratch）也依赖同一函数，需确保路径一致。此风险已基本解决。
  - 正确性风险：C++ 侧 maybe_allocate_tensor 曾使用 reshape，可能返回副本导致数据不一致，现已改用 view 并检查连续性。但需确保所有传入的 scratch 张量在该入口处都是连续的。
  - 性能风险：预分配缓冲区大小由 max_num_tokens 决定，若配置过大可能浪费显存，但通常由调度器控制，不会超过实际最大 batch。
  - 测试覆盖风险：仅增加了一个测试验证缓冲区复用，未覆盖所有组合（如 hidden_size 未提供时、不同 topk、不同专家数等），回归风险较小但存在。
- 影响：
  - 用户：小 batch 推理场景可获得 9-14% 的 kernel 加速，无需任何配置变更。大 batch 用户不受影响。

- 系统：显存分配次数减少，可能轻微降低显存碎片并提高分配效率。不影响模型加载时间或推断的数值精度。
- 团队：后续需要对 MoEPermuteScratch 数据结构保持向后兼容，新增 MoE 专家层实现时需遵循相同的 scratch 集成模式。测试和微基准已更新，可防止退化。
- 风险标记：平台兼容性，C++ 视图语义，测试覆盖不足

关联脉络

- 暂无明显关联 PR