

PR #52763 完整报告

vllm-project/vllm

[ROCM][CI] Attention test speedup

合并时间: 2026-08-19 05:40

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/52763>

执行摘要

- 一句话: 解锁 ROCm 注意力 kernel 测试, 整组提速 4.6 倍
- 推荐动作: 值得精读, 作为 pytest 测试性能优化的范本: 一是平台门控放函数体 (逐例空转 setup/teardown) 与放模块级 pytestmark (收集期一次性过滤) 的成本差异; 二是 skip_global_cleanup 的适用边界与保守选择方法论, PR body 对“为什么不标记 AITER/env 模块”的解释清晰。改动小、风险低、收益可量化, 适合快速合并; 后续维护需关注标记与模块行为演化的漂移。

功能与动机

PR body 指出两个相互独立的根因: 其一, test_merge_attn_states.py 以“only CUDA supports the custom merge_attn_states kernel”为由跳过全部 2592 个参数化用例, 但该 kernel 源码在构建时并未被 CUDA 门控, ROCm 同样编译注册且结果与 Triton 参考一致, 同文件的 both_empty 用例本就在 ROCm 上验证 native kernel; 其二, autouse 清理 fixture 在每个用例后 teardown 分布式环境并清空 allocator 缓存, 而纯 kernel 模块“there is nothing for that teardown to clean — but it still costs a garbage collection and a cache flush per test, which dominates the group's runtime”。本质是让 ROCm 测试资源花在真实 kernel 覆盖上, 而不是空转的 skip 与清理。

实现拆解

1. 问题识别与基线测量: 作者在同一块 MI300、相同 flags 与 GPU 条件下先跑 tests/kernels/attention 全量, 得到 46:34 基线, 定位两个互不相关的开销来源; 随后用默认环境二次确认, 排除测量 flags 的影响。
2. 批量挂载 skip_global_cleanup 标记 (commit 1, 158a0eed): 在 8 个纯 kernel 模块的 import 之后写入 pytestmark = pytest.mark.skip_global_cleanup, 涉及 test_attention.py、test_cache.py、test_cutlass_mla_decode.py、test_flashmla.py、test_prefix_prefill.py、test_triton_decode_attention.py、test_triton_unified_attention.py、test_triton_unified_attention_diffkv.py。这是测试套件既有的模块级机制, 语义明确: 模块不构建 engine、不初始化分布式状态, 跳过每用例后的全局 teardown, 省去一次垃圾回收与缓存刷新。
3. 重构 test_merge_attn_states.py 的平台门控 (commit 2, 68fd3c86): 把函数体内的 current_platform.is_cuda() 检查删除, 改为模块级 pytestmark 列表, 同时携带 skip_global_cleanup 与 skipif(not current_platform.is_cuda_alike(), ...).

is_cuda_alike() 同时接受 CUDA 与 ROCm，平台过滤在收集期一次性完成；ROCm 上 2592 个参数组合（含 fp8 变体，走 current_platform.fp8_dtype()）从空转 skip 变为真实 native kernel 覆盖。

4. 保守划定不动动的模块边界：ROCm 上全局清理还会刷新 AITER 类级状态，而 monkeypatch 无法恢复，因此所有涉及环境变量、AITER 或 patch 平台类的模块被刻意保留默认清理行为；作者说明最大模块已占整组运行时的大部分，保留这些模块不损失主要收益。
5. 验证与配套：无生产代码、配置、schema 或部署配套变更；本地单 MI300 全量对比加 test_merge_attn_states.py 单独对比 Triton 参考路径；CI 侧由维护者发起 /ci run，触发 Buildkite CI #84396。

测量结果（单块 MI300）：

指标	变更前	变更后
整组墙钟	46:34	10:08
通过用例	3207	5808
跳过用例	3703	1111
失败	0	0

关键文件：

- tests/kernels/attention/test_merge_attn_states.py（模块 注意力合并；类别 test；类型 test-coverage；符号 pytestmark, test_merge_attn_states）：核心变更文件：把平台门控从函数体内移到模块级 pytestmark，用 is_cuda_alike() 一次性接受 CUDA 与 ROCm，解锁 ROCm 上 2592 个参数化用例的真实 kernel 覆盖，是整组提速与新覆盖的关键。
- tests/kernels/attention/test_attention.py（模块 注意力；类别 test；类型 test-coverage；符号 pytestmark）：批量挂载 skip_global_cleanup 的 8 个文件之一，代表注意力 kernel 基准测试组；跳过无意义全局清理后每用例省去一次 GC 与缓存刷新。
- tests/kernels/attention/test_cache.py（模块 缓存层；类别 test；类型 test-coverage；符号 pytestmark）：缓存 kernel 测试模块，同样是空转全局清理的主要受害者之一，标记后随整组提速。
- tests/kernels/attention/test_cutlass_mla_decode.py（模块 MLA 解码；类别 test；类型 test-coverage；符号 pytestmark）：CUTLASS MLA decode 测试模块，含 compute capability 门控，标记 skip_global_cleanup 后免去无意义的每用例全局清理。
- tests/kernels/attention/test_flashmla.py（模块 MLA 算子；类别 test；类型 test-coverage；符号 pytestmark）：FlashMLA 解码测试模块，随整组跳过无内容的全局清理。
- tests/kernels/attention/test_prefix_prefill.py（模块 前缀预填；类别 test；类型 test-coverage；符号 pytestmark）：前缀预填 / chunked prefill kernel 测试模块，标记后整组收益一致。

- tests/kernels/attention/test_triton_decode_attention.py (模块 解码注意力; 类别 test; 类型 test-coverage; 符号 pytestmark) : Triton decode attention 测试模块, 属于纯 kernel 校验, 标记后跳过全局清理。
- tests/kernels/attention/test_triton_unified_attention.py (模块 统一注意力; 类别 test; 类型 test-coverage; 符号 pytestmark) : Triton unified attention 测试模块, 标记后整组提速收益一致。
- tests/kernels/attention/test_triton_unified_attention_diffkv.py (模块 差分注意力; 类别 test; 类型 test-coverage; 符号 pytestmark) : DiffKV 变体的 unified attention 测试模块, 随整组跳过无内容的全局清理。

关键符号: test_merge_attn_states

关键源码片段

tests/kernels/attention/test_merge_attn_states.py

核心变更文件: 把平台门控从函数体内移到模块级 pytestmark, 用 is_cuda_alike() 一次性接受 CUDA 与 ROCm, 解锁 ROCm 上 2592 个参数化用例的真实 kernel 覆盖, 是整组提速与新覆盖的关键。

```
import pytest
import torch

from vllm._custom_ops import merge_attn_states as merge_attn_states_cuda
from vllm.platforms import current_platform
from vllm.v1.attention.ops.triton_merge_attn_states import (
    merge_attn_states as merge_attn_states_triton,
)

# 模块级标记, 收集期即生效:
# 1. skip_global_cleanup —— 纯 kernel 测试不构建 engine、不初始化分布式状态,
# 跳过 autouse 全局清理 fixture, 省去每个用例的垃圾回收与缓存刷新;
# 2. skipif —— 平台门控从函数体内上移到模块收集期。
# 改用 is_cuda_alike() 同时接受 CUDA 与 ROCm: merge_attn_states 的 native
# 实现并未在构建时被 CUDA 门控, ROCm 同样编译注册, 结果与 Triton 参考一致。
pytestmark = [
    pytest.mark.skip_global_cleanup,
    pytest.mark.skipif(
        not current_platform.is_cuda_alike(),
        reason="merge_attn_states kernels require CUDA or ROCm.",
    ),
]

@pytest.mark.parametrize("use_fp8", [False, True])
@pytest.mark.parametrize("prefill_tokens_with_context", [None, 128])
# 其余参数组合: num_tokens / num_query_heads / head_size / input_dtype
@torch.inference_mode()
def test_merge_attn_states(
```

```

    prefill_tokens_with_context: int | None,
    num_tokens: int,
    num_query_heads: int,
    head_size: int,
    input_dtype: torch.dtype,
    use_fp8: bool,
):
    # 原实现将 current_platform.is_cuda() 检查放在函数体内:
    # ROCm 上 2592 个参数化用例仍要逐一执行 setup/teardown,
    # 花费数分钟却只是逐个 skip, 且掩盖了 ROCm 上 native kernel 的真实覆盖;
    # 该分支已删除, 平台判断在收集期一次性完成。
    NUM_TOKENS = num_tokens
    NUM_HEADS = num_query_heads
    HEAD_SIZE = head_size

    # use_fp8 时输入保持 input_dtype, 输出切换为平台 FP8 类型并带 per-tensor scale
    output_dtype = input_dtype
    output_scale = None
    if use_fp8:
        output_dtype = current_platform.fp8_dtype()
        output_scale = torch.tensor([0.05], dtype=torch.float32, device="cuda")
    # ... 后续 CUDA / Triton / PyTorch 三条路径的数值与性能对比逻辑保持不变

```

tests/kernels/attention/test_attention.py

批量挂载 skip_global_cleanup 的 8 个文件之一, 代表注意力 kernel 基准测试组; 跳过无意义全局清理后每用例省去一次 GC 与缓存刷新。

```

import random

import pytest
import torch

from tests.kernels.allclose_default import get_default_atol, get_default_rtol
from tests.kernels.utils import opcheck
from vllm import _custom_ops as ops
from vllm.model_executor.layers.attention import Attention, MMEncoderAttention
from vllm.platforms import current_platform
from vllm.utils.mem_utils import get_max_shared_memory_bytes
from vllm.utils.torch_utils import set_random_seed

# 该模块只校验 attention kernel, 不构建 engine、不初始化分布式状态,
# 因此复用测试套件既有的 skip_global_cleanup 机制,
# 跳过每个用例后的全局 teardown (释放缓存 + 垃圾回收)。
pytestmark = pytest.mark.skip_global_cleanup

FLOAT32_BYTES = torch.finfo(torch.float).bits // 8
# 根据共享内存大小推导可测的最大序列长度, 预留 512 字节 buffer
MAX_SEQ_LEN = get_max_shared_memory_bytes() // FLOAT32_BYTES - 512
# 任意测试值: 过大 NUM_BLOCKS 可能耗尽显存

```

```
NUM_BLOCKS = 4321
PARTITION_SIZE_ROCM = 256
DTYPES = [torch.bfloat16]
# 其余参数列表 (head sizes、block sizes、KV cache dtype 等) 保持不变
```

评论区精华

评审链路上实质讨论不多：claude[bot] 因 PR 来自 fork 而关闭自动审核，维护者 AndreasKaratzas 直接以 LGTM 批准并触发 /ci run。真正的设计权衡完整写在 PR body 里，包括“模块级门控优于函数内 skip 的成本差异”“为何保守选择不标记涉及 AITER/env 的模块”“为何用默认环境二次确认测量”，这些是比评论更有价值的决策记录。

- fork PR 自动审核关闭 (other): 未触发代码级自动审核，评审依赖维护者人工把关。
- 维护者批准与 CI 触发 (other): 合并放行，CI 流水线作为最终验证依据。

风险与影响

- 风险：
 1. skip_global_cleanup 标记漂移风险：标记基于“当前不建 engine、不初始化分布式状态”的事实；后续若这些模块引入依赖全局状态的用例（如分布式集合、跨用例共享缓存），会静默失去清理，出现难排查的污染，需在维护时留意。
 2. AITER 类级状态边界：ROCm 上全局清理会刷新 AITER 类级状态而 monkeypatch 不恢复，已标记模块若将来引入 env 或 AITER 操作，需重新评估标记是否仍安全。
 3. 解锁用例的平台覆盖有限：验证仅在单块 MI300 完成，gfx90a 等其他 ROCm 架构上 native kernel 与 fp8 变体的行为未实测，存在出现新平台性失败的可能。
 4. CI 提速未实测：PR 自述“the magnitude there is not measured here”，CI 多卡共享、不同驱动下的收益与稳定性需以实际流水线运行确认；本 PR 已触发 Buildkite #84396 作为放行依据。- 影响：生产功能零影响；ROCm 用户获得 merge_attn_states native kernel 的 2592 个参数组合（含 fp8）回归保护，弥补了此前“有 kernel 实现却无覆盖”的空白。团队与 CI 侧收益显著：单 MI300 墙钟 46:34 → 10:08（约 4.6 倍），通过用例 3207 → 5808，跳过 3703 → 1111；省去的是每用例垃圾回收与缓存刷新，成本不依赖硬件，CI 同样受益。改动范围仅 tests/kernels/attention 下 9 个测试文件，回滚成本低。- 风险标记：纯测试侧变更，解锁 2592 个新用例，覆盖仅单 MI300 验证，CI 提速幅度未实测，skip_global_cleanup 标记漂移风险

关联脉络

- PR #40938 [ROCm][CI] Move ROCm AITER quantization tests: 同为 ROCm 测试组织架构；本 PR 特意以 AITER 类级状态与 monkeypatch 不恢复为由，保守选择不标记相关模块，与 #40938 对 ROCm AITER 测试的治理线直接相关。
- PR #52659 [CI] Standardize test job labels by device: 同属 kernels.yaml 测试套件的 CI 组织优化，统一设备标签与本 PR 的按设备提速互为配套，共同改善 ROCm/NVIDIA kernel 测试的 CI 效率。

- PR #52810 [CI][ROCm] Prevent Git maintenance races during shallow fetches: 同期 ROCm CI 可靠性加固，本 PR 是同一波 ROCm CI 提速与加固投入的一部分，反映团队对 ROCm CI 稳定性和时延的持续关注。