

PR #54312 完整报告

vllm-project/vllm

[CI][Test] Deflake test_mem.py sleep-mode asserts via allocator bookkeeping

合并时间: 2026-08-30 03:05

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54312>

执行摘要

- 一句话: 分配器簿记替代设备级查询, 修复 sleep 测试 flaky
- 推荐动作: 值得快速精读, 重点学习两点: 一是对 torch.accelerator.get_memory_info() 语义的辨析——docstring 明说 free 值是 "available across all processes and applications", 任何基于该值增量的时序断言都天然脆弱; 二是 co-tenant A/B 实验设计, 用受控噪声量化 flake 根因, 而不是猜测。若团队要写类似的 allocator 测试, 可先确认生产对象是否已暴露类似 pointer_to_data 的可观测簿记。无需精读生产代码, 因为本 PR 不含任何生产逻辑变更。

功能与动机

PR body 明确指出: 三个测试断言 sleep()/discard() 释放了内存, 但所用仪器是错误的。torch.accelerator.get_memory_info() 的 docstring 说明 free 值是 "available across all processes and applications", 实现上直通 cudaMemGetInfo/hipMemGetInfo, 因此两次采样之间的差值携带了 GPU 上任意其他进程的分配 / 释放噪声。被测信号极小 (test_basic_cumem 睡眠 8 MiB, test_discard_tags 丢弃 1 MiB), 在 192 GiB 卡上 co-tenant 移动几十 MiB 就足以翻转断言。该 flake 在 AMD CI 的 MI300X 上于 PR #52707 (无关, 已合并) 的 CI 中复现, 失败时 allocator 日志显示 "sleep freed 0.02 GiB" 确实释放了, 但设备级读数仍走向反方向, 证明是测量工具问题而非分配器问题。

实现拆解

1. 新增模块级 helper `mapped_usage()`: 位于 tests/basic_correctness/test_mem.py 顶部。遍历 `allocator.pointer_to_data.values()`, 对 `not data.is_asleep` 的条目累加 `data.handle[1]`。AllocationData 是 vllm/device_allocator/init.py 中的共享 dataclass (含 handle、tag、cpu_backup_tensor、is_asleep), 因此 CuMemAllocator 与 XpuMemAllocator 可共用同一实现; `handle[1]` 是分配器按分配粒度向上取整后的尺寸。PR body 提到早期版本曾包含生产侧改动, 最终回退为纯测试方案, 避免为测试扩展生产 API。
2. 重写 `test_basic_cumem`: 删除两次 torch.accelerator.get_memory_info() 采样, 改为三段精确断言——sleep 前 `mapped_bytes >= y.nbytes + z.nbytes` (防止空池使断言形同虚设)、sleep 后 `mapped_usage == 0`、wake_up 后精确恢复 `mapped_bytes`。后两段是同一量的两次读数, 用精确相等; 首段因粒度取整用 `>=`。

3. 重写 `test_discard_tags`: 将 " 设备空闲内存严格增大 " 改为双向界——`discard('kv_cache')` 前后映射量差值 $\geq \text{kv.nbytes}$ (至少释放这么多), `discard` 后剩余 $\geq \text{weights.nbytes}$ (权重保持映射)。
4. 重写 `test_cumem_with_cudagraph`: 与 `test_basic_cumem` 相同模式, `sleep` 前 $\geq \text{weight.nbytes} + \text{cache.nbytes}$ 、`sleep` 后 $= 0$ 、`wake_up` 后精确恢复; CUDA graph 捕获与 `replay` 的原有功能检查保持不变。
5. 验证配套: 安静 GPU 上全文件 13 passed (515 秒); 在 8x MI355X 节点上用独立 co-tenant 进程循环分配 / 释放 128 MiB 做 A/B——pristine main 18 次运行 13 次失败、本分支 10 次全过; `ruff check/format (0.14.0)` 通过。

关键文件:

- `tests/basic_correctness/test_mem.py` (模块 内存测试; 类别 `test`; 类型 `test-coverage`; 符号 `mapped_usage`, `test_basic_cumem`, `test_discard_tags`, `test_cumem_with_cudagraph`): 唯一变更文件。新增 `mapped_usage()` 基于 `allocator` 的 `pointer_to_data` 簿记统计进程内映射字节, 替换 `device-wide` 的 `torch.accelerator.get_memory_info()` 采样, 重写 `test_basic_cumem`、`test_discard_tags`、`test_cumem_with_cudagraph` 三个测试的断言, 消除共享 GPU 上 co-tenant 噪声导致的 flaky。

关键符号: `mapped_usage`, `test_basic_cumem`, `test_discard_tags`, `test_cumem_with_cudagraph`

关键源码片段

`tests/basic_correctness/test_mem.py`

唯一变更文件。新增 `mapped_usage()` 基于 `allocator` 的 `pointer_to_data` 簿记统计进程内映射字节, 替换 `device-wide` 的 `torch.accelerator.get_memory_info()` 采样, 重写 `test_basic_cumem`、`test_discard_tags`、`test_cumem_with_cudagraph` 三个测试的断言, 消除共享 GPU 上 co-tenant 噪声导致的 flaky。

```
def mapped_usage(allocator) -> int:
    """统计分配器当前仍映射在设备上的字节数。
```

```
    不要用 `torch.accelerator.get_memory_info()` 作差分: 该调用是
    device-wide 的 `cudaMemGetInfo` / `hipMemGetInfo` 查询, 读数会
    混入同设备上其他进程的分配与释放噪声; 而分配器自己的
    `pointer_to_data` 簿记是进程本地且精确的, 才是我们真正想断言的状态。
    """
```

```
    return sum(
        data.handle[1] # handle[1] 是经分配粒度向上取整后的字节数
        for data in allocator.pointer_to_data.values()
        if not data.is_asleep # 只统计仍映射、未进入 sleep 状态的分配
    )
```

```
def test_basic_cumem():
```

```

# 默认内存池张量
shape = (1024, 1024)
x = torch.empty(shape, device=DEVICE_TYPE)
x.zero_()

# 自定义内存池张量
allocator = get_mem_allocator_instance()
with allocator.use_memory_pool():
    y = torch.empty(shape, device=DEVICE_TYPE)
    y.zero_()
    y += 1
    z = torch.empty(shape, device=DEVICE_TYPE)
    z.zero_()
    z += 2

# 各池张量仍可混用（原功能检查保持不变）
output = x + y + z
assert torch.allclose(output, torch.ones_like(output) * 3)

# 关键改动：断言分配器自身的映射量，而非设备级空闲内存。
# 在 sleep 前至少覆盖 y + z（避免空池使断言形同虚设）；
# sleep 后映射归零；wake_up 后精确恢复。handle[1] 带粒度
# 取整，故前一个断言用 >=；后两个是同一量的两次读数，可用
# 精确相等。
mapped_bytes = mapped_usage(allocator)
assert mapped_bytes >= y.nbytes + z.nbytes
allocator.sleep()
assert mapped_usage(allocator) == 0
allocator.wake_up()
assert mapped_usage(allocator) == mapped_bytes

# sleep/wake 后张量内容保持一致
output = x + y + z
assert torch.allclose(output, torch.ones_like(output) * 3)

```

评论区精华

本 PR 几乎没有实质性 review 交锋。claude[bot] 自动评论指出："This pull request is from a fork — automated review is disabled. A repository maintainer can comment [@claude review](#) to run a one-time review." 维护者 AndreasKaratzas 未触发额外审查，直接 APPROVED。最有价值的技术论证集中在 PR body 内部：作者先考证 `torch.accelerator.get_memory_info()` 的 docstring 与实现链（`at::getDeviceAllocator->getMemoryInfo` 直通驱动查询），证明它是 device-wide 读数；再用 co-tenant A/B 实验量化 flake（control 3 个可定位失败的 traceback 均落在同一断言，且 allocator 日志显示 "sleep freed 0.02 GiB" 与设备级读数反向），锁定 "测量工具错误而非分配器错误"。设计取舍上，作者明确解释 `>=` 与 `==` 的边界语义：`handle[1]` 带粒度向上取整，故跨量比较用 `>=`；sleep/wake 对比是同一量的两次读数，可用精确相等。

- fork PR 自动化 review 被禁用 (other): 维护者 AndreasKaratzas 未触发额外审查, 直接批准合并。

风险与影响

- 风险:
 - 与内部簿记强耦合: mapped_usage() 直接依赖 pointer_to_data、AllocationData.is_asleep、handle[1] 三个内部契约。若 vllm/device_allocator/init.py 或 cumem.py 未来重构这些字段, 测试会静默失真或报错, 需要同步跟进。
 - 断言信任分配器自我报告: 新断言验证的是分配器 " 声称映射了多少 ", 不再是驱动视角的真实空闲内存。若分配器簿记本身出错 (如 is_asleep 翻转遗漏、pointer_to_data 未更新), 测试可能跟着错, 失去对真实设备状态的兜底校验——原测试虽 flaky, 但至少测到了设备真实状态。
 - CUDA graph 覆盖有限: test_cumem_with_cudagraph 只断言分配器跟踪的映射量, CUDA graph 捕获产生的私有内存池是否进入 pointer_to_data 在材料中未明确, 该测试对 graph 内存的覆盖能力没有增强。
 - 平台可移植性: helper 对 XPU 同样有效 (共享 AllocationData), 但若未来新增分配器实现未遵循同一簿记结构, 测试将以 AttributeError 失败而非断言失败, 失败信息会变模糊。
- 影响:
 - CI 稳定性: 直接消除三个测试在共享 GPU 环境 (尤其 AMD MI300X/MI355X CI) 上的偶发失败, 不再要求测试节点独占设备。
 - 断言强度: 从 " 严格增大 " 升级为 " 精确相等 / 双向界 ", 能捕获分配器未释放映射的回归, 测试诊断价值反而提升。
 - 影响范围: 仅 tests/basic_correctness/test_mem.py 一个文件、+28/-9, 无生产代码、无用户可见行为变化。mapped_usage() 成为该文件可复用的测量模式, 后续 sleep/discard 测试可直接复用。
 - 团队: 为其他 GPU 内存测试提供了一个可复制的 " 进程内精确测量 " 范式, 避免再次误用 device-wide API 造成 flaky。
 - 风险标记: 测试与分配器内部簿记强耦合, 断言信任分配器自我报告, 粒度取整依赖 >= 语义, 缺真实设备状态兜底校验

关联脉络

- 暂无明显关联 PR