

PR #47404 完整报告

vllm-project/vllm

[ROCm] Synchronize sparse MLA metadata before graph replay

合并时间: 2026-07-10 03:59

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/47404>

执行摘要

- 一句话: 修复 ROCm sparse MLA 元数据同步竞态条件
- 推荐动作: 该 PR 值得精读。它展示了一个典型的 CUDA graph replay 下异步操作同步问题的修复模式: 在 graph-captured kernel 依赖的异步写入后插入 synchronize。测试用例的设计 (使用 mock 验证调用顺序) 也是良好实践, 适合作为异步同步单元测试的参考。

功能与动机

Issue #47196 报告在 ROCm gfx942 上使用 MTP 推测解码 + sparse MLA decode 并启用 CUDA graph 时, 出现 GPU memory access fault。根本原因是 `get_mla_metadata_v1()` 异步写入的持久化元数据缓冲区在 graph replay 时可能尚未完成, replay 读取到过时而部分更新的数据。该修复在元数据重计算后强制 stream 同步, 保证数据一致性。

实现拆解

1. 定位竞态条件: 在 `rocm_aiter_mla_sparse.py` 的 `ROCMAiterMLASparseMetadataBuilder.build()` 方法中, 当 `metadata_key` 变化时会调用 `get_mla_metadata_v1()` 异步计算持久化元数据 (如 `_mla_work_meta_data` 等缓冲区)。这些缓冲区随后被返回的 `ROCMAiterMLASparseMetadata` 对象引用, 并在 CUDA graph replay 时作为输入。但 write 未同步到 stream, replay 可能读到旧值。
2. 插入同步点: 在 `get_mla_metadata_v1()` 调用之后、`_prev_metadata_key` 更新之前, 添加一行 `torch.cuda.current_stream(self.device).synchronize()`。该同步只发生在真正重计算元数据时 (即 `metadata_key` 变化分支), 缓存命中路径不受影响。
3. 新增针对性测试: 创建 `tests/kernels/attention/test_rocm_aiter_mla_sparse_metadata_sync.py`, 通过 mock 替换 `aiter.get_mla_metadata_v1` 和 `torch.cuda.current_stream`, 验证 `build()` 的正确执行顺序: 第一次调用触发 metadata 重计算并跟随 synchronize, 第二次调用 (相同 `metadata_key`) 则既不计算也不同步。测试包含 ROCm 和 AITER 安装检测的模块级跳过条件。
4. CI 配置与兼容性: 测试文件只在 ROCm 且 AITER 可用时执行, 使用 `pytest.skip(allow_module_level=True)` 优雅跳过不兼容环境。

关键文件:

- `vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse.py` (模块 注意力; 类别 source; 类型 core-logic): 核心修复文件, 在 `ROCMAiterMLASparseMetadataBuilder.build()`

方法中 `get_mla_metadata_v1()` 调用后添加 `stream synchronize`。三行变更解决了CUDA graph replay 下持久化元数据未同步的竞态条件。

- `tests/kernels/attention/test_rocm_aiter_mla_sparse_metadata_sync.py` (模块 测试; 类别 `test`; 类型 `test-coverage`; 符号 `_FakeAiter`, `_make_builder`, `_make_common_metadata`, `test_sparse_persistent_metadata_syncs_only_after_recompute`) : 新增的针对性测试文件, 通过 `mock` 验证 `build()` 中元数据重计算和同步的正确顺序, 并测试缓存命中路径不会触发不必要的调用。包含 ROCm 和 AITER 跳过条件, 确保只在兼容环境中运行。

关键符号: `ROCMAiterMLASparseMetadataBuilder.build`,
`test_sparse_persistent_metadata_syncs_only_after_recompute`, `_make_builder`,
`_make_common_metadata`

关键源码片段

`vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse.py`

核心修复文件, 在 `ROCMAiterMLASparseMetadataBuilder.build()` 方法中 `get_mla_metadata_v1()` 调用后添加 `stream synchronize`。三行变更解决了 CUDA graph replay 下持久化元数据未同步的竞态条件。

```
# vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse.py
(ROCMAiterMLASparseMetadataBuilder.build)
```

```
# 当 metadata_key 变化时, 需要重新计算持久化元数据
if metadata_key != self._prev_metadata_key:
    from aiter import get_mla_metadata_v1
```

```
get_mla_metadata_v1(
    qo_indptr,
    paged_kv_indptr,
    paged_kv_last_page_len,
    self._num_attention_heads,
    1,
    True,
    self._mla_work_meta_data, # 持久化缓冲区, 将被 graph replay 读取
    self._mla_work_info_set,
    self._mla_work_indptr,
    self._mla_reduce_indptr,
    self._mla_reduce_final_map,
    self._mla_reduce_partial_map,
    page_size=1,
    kv_granularity=16,
    max_seqlen_qo=1,
    uni_seqlen_qo=1,
    fast_mode=True,
)
```

```
# 持久化元数据缓冲区由 graph replay 读取。
```

```
# 将异步元数据写入排序在 graph-captured decode kernel 之前。
```

```
torch.cuda.current_stream(self.device).synchronize()
self._prev_metadata_key = metadata_key
```

tests/kernels/attention/test_rocm_aiter_mla_sparse_metadata_sync.py

新增的针对性测试文件，通过 mock 验证 build() 中元数据重计算和同步的正确顺序，并测试缓存命中路径不会触发不必要的调用。包含 ROCm 和 AITER 跳过条件，确保只在兼容环境中运行。

```
# tests/kernels/attention/test_rocm_aiter_mla_sparse_metadata_sync.py

@torch.inference_mode()
def test_sparse_persistent_metadata_syncs_only_after_recompute(monkeypatch):
    # 构造一个 builder 实例（避免 __init__ 的复杂依赖）
    builder = _make_builder()
    common_metadata = _make_common_metadata()
    events: list[str] = []

    # fake 生成稀疏 seqlen
    def fake_generate_sparse_seqlen_triton(*args, **kwargs):
        return torch.tensor([1, 2], dtype=torch.int32, device="cpu")

    # fake aiter 模块，仅记录 metadata 调用
    fake_aiter = _FakeAiter("aiter")
    def fake_get_mla_metadata_v1(*args, **kwargs):
        events.append("metadata")
    fake_get_mla_metadata_v1_mock = Mock(side_effect=fake_get_mla_metadata_v1)
    fake_aiter.get_mla_metadata_v1 = fake_get_mla_metadata_v1_mock
    monkeypatch.setitem(sys.modules, "aiter", fake_aiter)
    monkeypatch.setattr(
        sparse_mod,
        "generate_sparse_seqlen_triton",
        fake_generate_sparse_seqlen_triton,
    )
    # 替换 torch.cuda.current_stream 返回带有 synchronize 的 fake stream
    monkeypatch.setattr(
        sparse_mod.torch.cuda,
        "current_stream",
        lambda device=None: SimpleNamespace(
            synchronize=lambda: events.append("sync")
        ),
    )

    # 第一次 build: metadata_key 变化，应触发 metadata 计算 + sync
    builder.build(common_prefix_len=0, common_attn_metadata=common_metadata)
    assert events == ["metadata", "sync"]
    assert fake_get_mla_metadata_v1_mock.call_count == 1

    # 第二次 build: metadata_key 相同，应无操作
    events.clear()
```

```
builder.build(common_prefix_len=0, common_attn_metadata=common_metadata)
assert events == []
assert fake_get_mla_metadata_v1_mock.call_count == 1
```

评论区精华

Review 主要集中在测试的健壮性：

- AndreasKaratzas要求为测试添加 ROCm 和 AITER 安装跳过条件（评论），zihaomu随后添加了模块级 skip。
- 一次 CI 失败（kernels-attention-test-2）由测试误在 CUDA 设备上创建张量导致，修复后通过（讨论）。
- 另一次 CI 失败被识别为 modelscope API 不兼容的无关问题（讨论）。
- subhajitdchow在评论中确认该修复在 8xMI300X 全规模场景下消除故障。
- 测试文件应添加 ROCm 和 AITER 跳过条件 (testing): zihaomu 在提交中增加了模块级 skip，使用 `pytest.skip(allow_module_level=True)` 跳过不兼容环境。
- CI 失败与 modelscope 兼容性问题无关 (other): 识别为无关失败，通过 rebase 解决。

风险与影响

- 风险：该变更仅在 `metadata_key` 变化分支中添加 stream synchronize，属于已有的重计算路径，不会影响缓存命中路径。同步操作本身可能引入微小的性能开销（单次同步延迟），但相对于 graph replay 后可能出现的 GPU fault，这是必要的 trade-off。风险极低，因为：
 - 仅影响 ROCm AITER sparse MLA 后端，不涉及其它注意力后端。
 - 不改变公共 API 或模型配置。
 - 单点插入，逻辑清晰，容易回滚。
- 影响：用户影响：修复了 ROCm 平台上使用 sparse MLA + MTP 推测解码 + CUDA graph 时可能遇到的 GPU 内存访问错误。用户无需任何配置更改即可受益。系统影响：无，仅增加一次 stream synchronize。团队影响：后续维护者需注意 ROCm 特定后端的异步操作顺序，尤其涉及 graph replay 的场景。
- 风险标记：竞态条件修复，CUDA graph 数据一致性，ROCm 特定路径

关联脉络

- 暂无明显关联 PR