

PR #54404 完整报告

vllm-project/vllm

[ROCm][CI] Add attention-sink support to ROCm AITER sparse MLA

合并时间: 2026-09-07 17:26

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54404>

执行摘要

- 一句话: 为 ROCm AITER 稀疏 MLA 添加注意力 sink 支持, 新增 LSE 解码算子和完整的回退 / 验证路径。
- 推荐动作: 该 PR 是一个重要且高质量的贡献, 解决了 ROCm 平台上的一个关键功能缺失。
建议:
 1. 值得精读: 对于关注 ROCm 支持、注意力机制实现或 AITER 库集成的工程师, 该 PR 值得仔细阅读, 特别是 `_forward_mla` 中的路径选择逻辑和 `_rocm_aiter_mla_decode_fwd_lse_impl` 中的硬件特定处理。
 2. 关注设计决策: 作者在 `backend.py` 中引入 `supports_dcp()` 检查是一个有意义的架构改进, 尽管可能产生意外后果。建议关注后续的默认值调整 PR。
 3. 评估测试质量: 测试套件非常全面, 使用了双精度参考和精度预算来验证数值正确性, 这种做法值得在其他地方推广。
 4. 注意后续工作: 该 PR 包含一些技术债务 (如代码重构建议) 和临时措施 (如跳过的测试), 需要关注后续的清理工作。

功能与动机

本 PR 的核心动机是为在 PR#54160 中引入的可学习注意力 sink 功能提供 ROCm 平台的实现支持。正如 PR body 所述: 'HY-V4 introduced learnable attention sinks in #54160, but its ROCm initialization was unsupported in AMD build 12635 and build 12653.' 该 PR 是模型启用 PR#54405 的必要前置条件。此外, 本地 MI300 验证还暴露了空行 NaN、错误拒绝 FP16 输入以及缺少原生 BF16 内核 (64 头桶) 等问题, 这些都需要在此次变更中修复。

实现拆解

1. 新增非持久化 LSE 解码算子: 在 `vllm/_aiter_ops.py` 中, 新增了 `_rocm_aiter_mla_decode_fwd_lse_impl` 和 `_rocm_aiter_mla_decode_fwd_lse_fake` 两个函数, 并通过 `direct_register_custom_op` 将它们注册为 `rocm_aiter_mla_decode_fwd_lse` 自定义算子。该算子明确设计为不接受持久化元数据 (`work_meta_data`), 以解决 `gfx942` 持久化内核不提供 LSE 代码对象的问题。算子内部包含针对 `gfx950 H32 BF16` 和 `gfx942 FP8` 的特殊分片处理逻辑。
2. 扩展稀疏 MLA 后端以支持 sink: 在 `vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse.py` 中, `ROCMAiterMLASparseBackend` 类新增了 `supported_kv_cache_layouts()` 和

`supports_sink()` 类方法。ROCMAiterMLASparseImpl 的 `__init__` 方法现在接受并验证 `sinks` 参数（要求为 float32、1 维、连续）。`_forward_mla` 方法的返回类型从 `torch.Tensor` 变为 `tuple[torch.Tensor, torch.Tensor | None]`，并新增了复杂的路径选择逻辑：当存在 sink 且 AITER 不支持对应数据类型 / 头数时，回退到 Triton 路径；否则调用新增的 `mha_decode_fwd_lse` 算子。

3. 增强后端能力发现与验证：在 `vllm/v1/attention/backend.py` 的基类 `AttentionBackend` 中，新增了 `supports_dcp()` 类方法，并在 `validate_configuration` 中增加了对 `use_dcp` 的检查。这填补了之前对 DCP（解码上下文并行）支持检查的空白，尽管可能意外地影响了其他后端（如 Triton MLA）。
4. 添加全面的测试覆盖：新增了多个测试文件：`tests/kernels/attention/test_rocm_aiter_mla_sink.py`（包含与双精度参考比较的精确度测试、数据类型拒绝测试、元数据验证测试等）、`tests/kernels/attention/test_rocm_aiter_mla_op_registration.py`（新增 LSE 算子 schema 测试）、`tests/kernels/attention/test_rocm_aiter_mla_sparse_metadata_sync.py`（新增 sink 构建跳过持久化元数据的测试）。同时修改了 `tests/v1/attention/test_rocm_glm5next_sparse.py` 以测试 sink 在 Triton 路径中的保留情况，以及 `tests/models/test_initialization.py` 临时跳过 ROCm 上的 HY-V4 初始化测试。
5. 修复基础设施与输出逻辑：在 `vllm/v1/attention/ops/rocm_aiter_mla_sparse.py` 中，修复了 Triton 预填充内核的输出数据类型和形状处理问题。

关键文件：

- `vllm/_aiter_ops.py`（模块 自定义算子；类别 source；类型 core-logic；符号 `_rocm_aiter_mla_decode_fwd_lse_impl`, `_rocm_aiter_mla_decode_fwd_lse_fake`, `mha_decode_fwd_lse`）：新增了 `rocm_aiter_mla_decode_fwd_lse` 自定义算子及其 fake 实现，是 sink 功能在 AITER 层面的核心入口。
- `vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse.py`（模块 注意力后端；类别 source；类型 core-logic；符号 `supported_kv_cache_layouts`, `supports_sink`, `ROCMAiterMLASparseImpl`）：稀疏 MLA 后端的核心实现，新增了 sink 支持、DCP 检查、路径选择逻辑和数据类型验证。
- `tests/kernels/attention/test_rocm_aiter_mla_sink.py`（模块 Sink 测试；类别 test；类型 test-coverage；符号 `_sink_reference`, `_assert_sink_output_close`, `test_sparse_mla_sink_matches_ragged_reference`）：新增的核心测试文件，全面验证 sink 功能的数值正确性、元数据验证和错误处理。

关键符号：`_rocm_aiter_mla_decode_fwd_lse_impl`, `_rocm_aiter_mla_decode_fwd_lse_fake`, `ROCMAiterMLASparseBackend.supported_kv_cache_layouts`, `ROCMAiterMLASparseBackend.supports_sink`, `ROCMAiterMLASparseImpl.init`, `ROCMAiterMLASparseImpl._forward_mla`, `AttentionBackend.supports_dcp`

关键源码片段

`vllm/_aiter_ops.py`

新增了 `rocm_aiter_mla_decode_fwd_lse` 自定义算子及其 fake 实现，是 sink 功能在 AITER 层面的核心入口。

```
def _rocm_aiter_mla_decode_fwd_lse_impl(
    q: torch.Tensor,
    kv_buffer: torch.Tensor,
    o: torch.Tensor,
    qo_indptr: torch.Tensor,
    max_seqlen_qo: int,
    kv_indptr: torch.Tensor | None = None,
    kv_indices: torch.Tensor | None = None,
    kv_last_page_lens: torch.Tensor | None = None,
    sm_scale: float = 1.0,
    logit_cap: float = 0.0,
    q_scale: torch.Tensor | None = None,
    kv_scale: torch.Tensor | None = None,
) -> torch.Tensor:
    """Run non-persistent AITER MLA decode and return natural-log LSE.

    gfx942 persistent MLA kernels do not provide an LSE code object. Keeping
    this as a separate op makes that constraint structural: callers that need
    LSE cannot accidentally forward persistent work metadata.
    """

    from aiter.mla import get_meta_param, mla_decode_fwd

    kwargs: dict[str, float | int | torch.Tensor | None | bool] = {
        "sm_scale": sm_scale,
        "logit_cap": logit_cap,
        "return_lse": True,
    }
    if _check_aiter_mla_fp8_support():
        kwargs["q_scale"] = q_scale
        kwargs["kv_scale"] = kv_scale

    if (
        q.dtype == torch.bfloat16
        and kv_buffer.dtype == torch.bfloat16
        and q.shape[1] == 32
    ):
        from vllm.platforms.rocm import on_gfx950

        if on_gfx950():
            # The gfx950 H32 BF16 kernel mishandles ragged tails with split KV.
            # Its single-split path returns correct output and natural-log LSE.
            kwargs["num_kv_splits"] = 1
            kwargs["num_kv_splits_indptr"] = torch.arange(
                qo_indptr.shape[0], dtype=torch.int32, device=q.device
            )

    if q.dtype == FP8_DTYPE:
```

```

assert kv_indices is not None
num_kv_splits, num_kv_splits_indptr = get_meta_param(
    None,
    qo_indptr.shape[0] - 1,
    kv_indices.numel(),
    q.shape[1],
    max_seqlen_qo,
    q.dtype,
)
if num_kv_splits == 1:
    # gfx942's one-split FP8 asm writes the final output directly but
    # does not write either LSE buffer. Force the normal split reducer,
    # which produces both the same output and an accurate natural LSE.
    num_kv_splits = 2
    num_kv_splits_indptr = torch.arange(
        0,
        (qo_indptr.shape[0]) * num_kv_splits,
        num_kv_splits,
        dtype=torch.int32,
        device=q.device,
    )
kwargs["num_kv_splits"] = num_kv_splits
kwargs["num_kv_splits_indptr"] = num_kv_splits_indptr

_, final_lse = mla_decode_fwd(
    q,
    kv_buffer.view(-1, 1, 1, q.shape[-1]),
    o,
    qo_indptr,
    kv_indptr,
    kv_indices,
    kv_last_page_lens,
    max_seqlen_qo,
    **kwargs,
)
if final_lse is None:
    raise RuntimeError("AITER MLA decode did not return the requested LSE")
return final_lse

```

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

稀疏 MLA 后端的核心实现，新增了 sink 支持、DCP 检查、路径选择逻辑和数据类型验证。

```

class ROCMAiterMLASparseImpl(MLAAttentionImpl[ROCMAiterMLASparseMetadata]):
    is_sparse = True
    supports_dense_mha_prefill = False
    supports_dcp = False # 显式声明不支持 DCP

    def __init__(
        self,

```

```

num_heads: int,
head_size: int,
scale: float,
# ... 其他参数 ...
sinks: torch.Tensor | None = None,
# ... 其他参数 ...
):
    # ... 初始化逻辑 ...
    sinks = mla_args.pop("sinks", None)
    if sinks is not None:
        # 验证 sink 参数: 必须是 float32、1 维、连续
        if sinks.dtype != torch.float32:
            raise ValueError(
                f"ROCm AITER MLA sinks must be float32, got {sinks.dtype}"
            )
        if sinks.ndim != 1 or sinks.numel() != num_heads:
            raise ValueError(
                "ROCm AITER MLA sinks must have shape "
                f"({num_heads},), got {tuple(sinks.shape)}"
            )
        if not sinks.is_contiguous():
            raise ValueError("ROCm AITER MLA sinks must be contiguous")
    self.sinks: torch.Tensor | None = sinks
    # ... 其他初始化 ...

def _forward_mla(
    self,
    layer,
    q: torch.Tensor, # [sq, heads, d_qk]
    kv_c_and_k_pe_cache: torch.Tensor, # [blocks, heads, d_qk]
    attn_metadata: ROCMAiterMLASparseMetadata,
) -> tuple[torch.Tensor, torch.Tensor | None]:
    # 返回类型改为 (output, lse)
    # ...
    need_lse = self.sinks is not None
    triton_sink_fallback = (
        need_lse
        and q.dtype == kv_c_and_k_pe_cache.dtype
        and (
            q.dtype == torch.float16
            or (
                q.dtype == torch.bfloat16
                and (
                    mla_num_heads > 128
                    or (on_gfx942()) and 32 < mla_num_heads <= 64
                )
            )
        )
    )
    )
)

```

```
if triton_sink_fallback or _use_rocm_sparse_triton(...):
    # 走 Triton 回退路径
    # ...
    return output, None
# 否则走 AITER LSE 路径
# ... 验证数据类型和头数 ...
lse = rocm_aiter_ops.mla_decode_fwd_lse(...)
return output, lse
```

评论区精华

Review 讨论主要集中在以下几个方面：

- DCP 支持检查的影响范围：reviewer [tjtanaa](#) 指出，在 `backend.py` 中新增 `supports_dcp()` 方法可能影响所有其他后端，包括本身支持 DCP 的 Triton MLA 后端。作者 [AndreasKaratzas](#) 承认了这一问题，并确认 `supports_dcp` 在基类实现中默认返回 `True`（通过 `get_impl_cls().supports_dcp`），因此 Triton MLA 后端不受影响。作者计划在后续 PR 中将默认值改为 `False` 并显式启用现有 DCP 实现。审批者 [dllehr-amd](#) 也表达了类似担忧，但最终认为这是正确的方向。
- 代码重构建议：reviewer [tjtanaa](#) 建议重构 `_rocm_aiter_mla_decode_fwd_impl` 和 `_rocm_aiter_mla_decode_fwd_lse_impl`，提取公共逻辑并创建包装器以区分就地（inplace）和非就地（outofplace）操作。作者同意了这一建议，但出于避免合并冲突和 CI 运行成本的考虑，将其推迟到后续清理工作中。
- AI 代码审查意见：[coderabbitai\[bot\]](#) 提出了一些具体的改进点，包括：1) 在 `_rocm_aiter_mla_decode_fwd_lse_impl` 中，将 FP8 的两分片回退限制在 `gfx942`，以避免在 `gfx950` 上进行不必要的分割归约工作；2) 在 `rocm_aiter_mla_sparse.py` 中，将 `gfx942` 不支持的 BF16 sink 配置验证提前到 `__init__` 阶段，以防止引擎在 CUDA 图捕获期间启动失败。这些意见尚未被解决。
 - DCP 支持检查的影响范围 (design): 作者确认了当前实现不会影响现有后端，但承认这是一个需要后续改进的架构点。
 - 解码算子代码重复 (design): 代码重构被接受为后续工作项，当前 PR 优先合并。
 - FP8 两分片回退应限于 `gfx942` (performance): 未解决。该优化可能在后续 PR 中实现。
 - `gfx942` 不支持的 sink 配置应提前验证 (correctness): 未解决。可能在后续 PR 中改进验证时机。

风险与影响

- 风险：
 1. DCP 支持检查的兼容性风险：在 `backend.py` 中新增的 `supports_dcp()` 方法可能影响其他后端的配置验证。虽然作者确认基类默认实现为 `True`（通过 `get_impl_cls().supports_dcp`），因此当前不受影响，但这一变更引入了一个新的验证检查点。如果未来有后端未正确实现此属性，可能会导致意外的行为。此外，作者计划在后续 PR 中将默认值改为 `False`，这需要确保所有支持 DCP 的后端都得到显式启用。

1. 硬件特定逻辑的维护风险: `_rocm_aiter_mla_decode_fwd_lse_impl` 和 `rocm_aiter_mla_sparse.py` 中包含针对 gfx942、gfx950 以及特定头数（如 H32、H64）的条件逻辑。这些硬编码的硬件特定路径增加了代码的复杂性，并在未来新硬件发布时可能需要调整。
2. 数据类型与头数组合验证的边界条件: `_forward_mla` 中的数据类型和头数验证逻辑较为复杂，涉及多个条件分支。如果组合情况处理不周，可能导致运行时错误或静默的数值错误。尽管测试覆盖了主要场景，但一些边缘情况（如混合数据类型或特定头数与硬件的组合）可能未被完全覆盖。
3. 临时跳过的测试: `tests/models/test_initialization.py` 中对 ROCm 上 HY-V4 初始化的跳过是临时的，依赖于后续 PR#54405 的完成。如果该后续 PR 延迟或失败，这些测试将保持跳过状态，可能掩盖潜在问题。- 影响：对用户的影响：
 - 正面影响：该 PR 解决了 HY-V4 模型在 AMD ROCm GPU（MI300/MI350）上无法初始化的关键问题，使 ROCm 用户能够使用带有可学习注意力 sink 的 HY-V4 模型。这显著扩展了 vLLM 在 AMD 硬件上的功能支持。
 - 负面 / 中性影响：由于变更集中在 ROCm 后端，对 NVIDIA CUDA 用户没有直接影响。但是，`backend.py` 中的 `supports_dcp()` 检查可能影响所有后端的配置验证逻辑，需要关注。

对系统的影响：

- 性能：新增的 LSE 解码算子和 sink 应用逻辑引入了额外的计算开销，但这是实现 sink 功能所必需的。通过 Triton 回退路径和持久化元数据缓存机制，系统尽可能地保持了性能。
- 正确性：通过引入 FP64 参考实现和基于精度预算的验证，显著提升了数值正确性的保证。修复了空行 NaN 和 FP16 输入被错误拒绝的问题。
- 可维护性：新增的硬件特定逻辑和复杂的路径选择增加了代码的维护成本。需要确保未来的硬件支持能够清晰地整合到现有框架中。

对团队的影响：

- 开发流程：该 PR 是 ROCm 上 HY-V4 模型启用系列的一部分（与 PR#54405 配套）。它建立了一个模式，即为新的注意力特性（如 sink）提供平台特定的实现。
- 测试基础设施：新增的全面测试套件为 ROCm AITER MLA 注意力后端建立了高标准，可能作为未来类似工作的参考。
- 风险标记：硬件特定逻辑维护成本，DCP 检查潜在兼容性影响，临时测试跳过，复杂路径选择逻辑

关联脉络

- PR #54405 [ROCM][CI] Enable HY-V4 model initialization on ROCm: 本 PR 是 PR#54405 的直接前置依赖，为其提供了必要的 sink 支持。两个 PR 共同完成了 ROCm 上 HY-V4 模型的完整启用。
- PR #54160 [Feature] Add learnable attention sinks for HY-V4: 本 PR 中实现的 sink 功能最初在 PR#54160 中为 CUDA 引入。本 PR 将该功能扩展到 ROCm 平台。

- PR #54594 [ROCm] Validate HY-V4 on MI350X (gfx950): 独立验证 PR，使用了本 PR 的代码变更在 MI350X 硬件上进行了端到端测试，确认了功能的正确性。