

PR #54855 完整报告

vllm-project/vllm

[ROCm][Perf] Route large DSV4 sparse prefill to AITER OPUS

合并时间: 2026-09-10 14:39

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/54855>

执行摘要

- 一句话: 为 ROCm 上 DeepSeek-V4 稀疏 MLA 预填充引入 AITER OPUS 内核路由, 提升大请求量吞吐。
- 推荐动作: 这是一个高质量、低风险的性能优化 PR。强烈建议精读, 原因如下: 1) 它展示了如何在 vLLM 中安全地引入硬件特定优化内核的模式——通过严格的条件守卫、清晰的回退路径和全面的测试。2) 讨论中关于性能阈值的决策过程体现了数据驱动的技术判断, 对后续类似工作有借鉴意义。3) 代码变更集中, 逻辑清晰, 但涉及底层张量操作和内核调度, 理解其细节有助于把握 ROCm 上 MLA 注意力机制的性能优化方向。

功能与动机

旨在解决 DeepSeek-V4 模型在 ROCm 平台上, 大请求量稀疏 MLA 预填充的性能瓶颈。PR body 中的数据显示, 当最大并发数为 48 时, 使用 OPUS 内核可将总吞吐提升约 4.96%, 且评估结果 (gsm8k) 保持不变, 证明了性能提升未牺牲正确性。

实现拆解

1. 新增 OPUS 内核获取与路由条件判断: 在 `vllm/v1/attention/ops/rocm_aiter_mla_sparse.py` 中, 新增了 `_get_aiter_sparse_prefill_opus` 函数, 用于条件性导入并缓存 AITER OPUS 内核。同时, 新增了 `_can_use_aiter_sparse_prefill_opus` 函数, 作为路由守卫, 其判断逻辑严格, 检查了硬件平台 (`on_gfx950`)、最小查询数阈值、张量布局 (`stride`)、数据类型、设备以及 `attn_sink` 的一致性。这确保了只有在硬件和输入均满足要求时才会尝试使用 OPUS。
2. 实现 OPUS 内核调用与稠密到稀疏索引转换: 新增了 `_rocm_sparse_attn_prefill_ragged_aiter_opus` 函数, 作为 OPUS 内核的实际调用包装器。该函数负责将输入的稠密稀疏索引 (`indices`) 转换为 AITER OPUS 所需的 `ragged` 格式 (`ragged_indices` 和 `indptr`)。此转换逻辑被封装在函数内部, 与回退路径保持一致。
3. 修改主预填充入口点的调度逻辑: 在核心的 `rocm_sparse_attn_prefill` 函数中, 插入了 OPUS 路由分支。该逻辑首先检查 `rocm_aiter_mla_sparse` 模块级的 `_can_use_aiter_sparse_prefill_opus` 和 `_get_aiter_sparse_prefill_opus` 是否满足条件。若满足, 则尝试调用 `_rocm_sparse_attn_prefill_ragged_aiter_opus`。如果调用成功则直接返回; 否则, 流程会自然回退到原有的 Triton 实现。这种设计是纯粹的增量式优化, 保证了向后兼容。

4. 添加详尽的测试覆盖：在 `tests/kernels/attention/test_rocm_triton_attn_dsv4.py` 中，新增了超过 200 行测试用例，全面覆盖了新的路由逻辑。测试内容包括：对路由条件的参数化单元测试（如查询数量阈值、平台检查）；验证不兼容输入（如错误 stride、CPU 张量）被正确拒绝；通过 monkeypatch 模拟 `fake_opus` 和 `fake_dense_fallback` 来测试路由决策的正确性；以及一个在真实硬件（`requires_gfx950`）上验证 OPUS 内核计算正确性的端到端测试。

关键文件：

- `vllm/v1/attention/ops/rocm_aiter_mla_sparse.py`（模块 注意力算子；类别 `infra`；类型 `core-logic`；符号 `_get_aiter_sparse_prefill_opus`, `_can_use_aiter_sparse_prefill_opus`, `_rocm_sparse_attn_prefill_ragged_aiter_opus`, `rocm_sparse_attn_prefill`）：核心变更文件，实现了 OPUS 内核的路由条件判断、实际调用逻辑和在主预填充函数中的集成点。
- `tests/kernels/attention/test_rocm_triton_attn_dsv4.py`（模块 注意力测试；类别 `test`；类型 `test-coverage`；符号 `test_aiter_sparse_prefill_opus_selection`, `test_aiter_sparse_prefill_opus_selection_rejects_incompatible_inputs`, `test_sparse_attn_prefill_aiter_opus_routing`, `test_sparse_attn_prefill_preserves_dense_triton_fallback`）：为新增的 OPUS 路由逻辑提供了全面、分层的测试覆盖，是验证此性能优化安全性和正确性的关键。

关键符号：`_get_aiter_sparse_prefill_opus`, `_can_use_aiter_sparse_prefill_opus`, `_rocm_sparse_attn_prefill_ragged_aiter_opus`, `rocm_sparse_attn_prefill`, `test_aiter_sparse_prefill_opus_selection`, `test_sparse_attn_prefill_aiter_opus_routing`, `test_sparse_attn_prefill_ragged_aiter_opus`

关键源码片段

`vllm/v1/attention/ops/rocm_aiter_mla_sparse.py`

核心变更文件，实现了 OPUS 内核的路由条件判断、实际调用逻辑和在主预填充函数中的集成点。

```
# 条件性导入并缓存 AITER OPUS 内核
@functools.cache
def _get_aiter_sparse_prefill_opus() -> Callable[..., torch.Tensor] | None:
    from vllm._aiter_ops import rocm_aiter_ops

    # 仅当 MLA 功能启用时尝试导入
    if not rocm_aiter_ops.is_mla_enabled():
        return None
    try:
        from aiter.ops.pa_sparse_prefill_opus import pa_sparse_prefill_opus
    except ImportError:
        return None
    logger.info_once("Using AITER OPUS for large sparse MLA prefill on gfx950")
    return pa_sparse_prefill_opus

# 查询数量性能交叉点阈值（TP8 基准测试得出）
_GFX950_AITER_SPARSE_PREFILL_OPUS_MIN_QUERIES = 1024
```

```

# 判断是否可以使用 AITER OPUS 内核的严格守卫函数
def _can_use_aiter_sparse_prefill_opus(
    q: torch.Tensor,
    kv: torch.Tensor,
    attn_sink: torch.Tensor | None,
    output: torch.Tensor,
    on_gfx950: bool = _ON_GFX950,
) -> bool:
    return (
        on_gfx950
        and q.shape[0] >= _GFX950_AITER_SPARSE_PREFILL_OPUS_MIN_QUERIES # 性能阈值
        and q.is_cuda
        and output.shape == q.shape
        and kv.shape[-1] == q.shape[-1] # head_dim 一致
        and q.dtype in (torch.bfloat16, torch.float16) # 支持的数据类型
        and kv.dtype == q.dtype
        and output.dtype == q.dtype
        and kv.device == q.device
        and output.device == q.device
        and q.stride(-1) == 1 # 内存布局要求
        and kv.stride(-1) == 1
        and output.stride() == q.stride()
        and attn_sink is not None # attn_sink 必须存在
        and attn_sink.shape == (q.shape[1],) # 形状匹配
        and attn_sink.dtype == torch.float32
        and attn_sink.device == q.device
    )

# 在主预填充函数中插入的 OPUS 路由逻辑片段
# opus_attn_sink = None if attn_sink is None else attn_sink[: q.shape[1]]
# if (
#     _can_use_aiter_sparse_prefill_opus(q, kv.squeeze(1), opus_attn_sink, output)
#     and _get_aiter_sparse_prefill_opus() is not None
# ):
#     # ... 转换索引并尝试调用 OPUS ...
#     if _rocm_sparse_attn_prefill_ragged_aiter_opus(...):
#         return # 成功则提前返回，否则继续回退到 Triton

```

tests/kernels/attention/test_rocm_triton_attn_dsv4.py

为新增的 OPUS 路由逻辑提供了全面、分层的测试覆盖，是验证此性能优化安全性和正确性的关键。

```

# 参数化测试：验证路由条件的选择边界
@pytest.mark.parametrize(
    ("num_queries", "on_gfx950", "expected"),
    [(1023, True, False), (1024, True, True), (1024, False, False)],
)
def test_aiter_sparse_prefill_opus_selection(
    num_queries: int, on_gfx950: bool, expected: bool

```

```

) -> None:
    from vllm.v1.attention.ops.rocm_aiter_mla_sparse import (
        _can_use_aiter_sparse_prefill_opus,
    )

    device = torch.device("cuda")
    q = torch.empty(num_queries, 16, HEAD_DIM, dtype=torch.bfloat16, device=device)
    kv = torch.empty(num_queries, HEAD_DIM, dtype=torch.bfloat16, device=device)
    attn_sink = torch.empty(16, dtype=torch.float32, device=device)
    output = torch.empty_like(q)

    # 断言函数在不同输入下返回预期的布尔值
    assert (
        _can_use_aiter_sparse_prefill_opus(
            q, kv, attn_sink, output, on_gfx950=on_gfx950
        )
        is expected
    )

# 测试在 OPUS 内核可用时，路由是否正确选择它
# 使用 monkeypatch 模拟内核和依赖函数
def test_sparse_attn_prefill_aiter_opus_routing(monkeypatch) -> None:
    from vllm.v1.attention.ops import rocm_aiter_mla_sparse as mod

    q = torch.empty(2, 1, HEAD_DIM, dtype=torch.bfloat16)
    # ... 设置测试输入 ...

    def fake_opus(*args, out):
        nonlocal opus_calls
        opus_calls += 1
        # ... 验证传入 OPUS 的参数 ...
        out.zero_()
        return out

    # 将所有外部依赖替换为模拟对象
    monkeypatch.setattr(mod, "_can_use_aiter_sparse_prefill_opus", lambda *args: True)
    monkeypatch.setattr(mod, "_get_aiter_sparse_prefill_opus", lambda: fake_opus)
    monkeypatch.setattr(
        mod,
        "_rocm_sparse_attn_prefill_triton",
        lambda *args, **kwargs: pytest.fail("unexpected dense Triton fallback"),
    )
    # ... 调用主函数并断言 OPUS 被调用 ...
    mod.rocm_sparse_attn_prefill(...)
    assert opus_calls == 1

```

评论区精华

核心讨论集中在确定 OPUS 内核的启用阈值上。审阅者 [shen-shanshan](#) 质疑了常量 `_GFX950_AITER_SPARSE_PREFILL_OPUS_MIN_QUERIES = 1024` 的设定依据，要求提供性能对比数据。作者 [jiacao-amd](#) 回复了详细的算子级 A/B 测试数据表，清晰地展示了 Triton 和 OPUS 在不同查询数量下的性能曲线，并指出了两者在查询数为 1024 附近存在性能交叉点，从而论证了该阈值的合理性。此讨论最终以作者提供了充分证据而得到结论。代码提交历史也显示，作者随后将阈值解释注释补充到了代码中（commit 996051bb）。

- OPUS 内核启用的查询数量阈值设定依据 (performance): 讨论以作者提供充分性能数据结束，阈值 1024 被接受为基于实测的性能优化门限，而非正确性限制。

风险与影响

- 风险：
 1. 兼容性风险：变更仅限于 ROCm 的 gfx950 平台，且 aiter 模块或 OPUS 内核不可用时会静默回退到 Triton，因此对非目标平台或旧版本 aiter 无影响。风险较低。
 2. 性能回退风险：引入了运行时条件检查和潜在的索引格式转换（稠密到 ragged），可能为不满足条件的小请求增加微小开销。但路由条件本身计算轻量，且只有满足条件的请求才会进入转换逻辑，影响有限。
 3. 正确性风险：这是最高风险点。OPUS 内核被正确路由后，其计算结果的正确性至关重要。PR 通过一个使用 monkeypatch 模拟的路由逻辑测试（`test_sparse_attn_prefill_aiter_opus_routing`）和一个在真实硬件上运行的端到端正确性测试（`test_sparse_attn_prefill_ragged_aiter_opus`）来缓解此风险。测试使用了高容差（`atol/rtol=2e-2`）进行断言，这可能意味着 OPUS 内核在浮点精度上与参考实现存在微小差异，但应在可接受范围内。
 4. 索引转换正确性风险：稠密索引到 ragged 索引的转换逻辑是新增的关键环节。测试通过 mock 验证了转换函数的调用，并确保在回退路径中不会意外触发。但更全面的端到端测试可能依赖于真实 AITER 内核的运行。
 - 影响：直接影响：仅影响在 AMD gfx950 (MI355X) 硬件上运行 DeepSeek-V4 模型、且稀疏 MLA 预填充查询数量 ≥ 1024 的请求路径。此类请求的吞吐量将获得显著提升（最高约 5%）。系统影响：作为内核路由优化，其影响是局部的，不影响调度器、前端或模型定义等其他子系统。通过 `logger.info_once` 在首次使用时记录 OPUS 启用状态，增加了可观测性。团队影响：为 ROCm 团队提供了利用厂商特定优化内核的实践范例。详尽的测试用例为后续类似优化（如其他内核的路由选择）提供了参考。性能数据的讨论也加强了团队对于通过基准测试驱动决策的实践。
- 风险标记：平台特定优化（gfx950），性能阈值需根据硬件 / 配置调整，依赖外部 AITER 库

关联脉络

- PR #55887 [ROCm][Bugfix] Support shared KV prefill in AITER attention: 此 PR 修复了 AITER 注意力后端在共享 KV 缓存场景下的问题，而本 PR 是在同一 AITER 后端上进行的性能优化，两者共同完善了 ROCm 平台上的 AITER 注意力支持。
- PR #52664 [Performance][ROCm] Integrate aiter indexer scoring and top-k kernels into MiniMax-M3 sparse attention path: 此 PR 展示了在 MiniMax-M3 模型上集成 AITER 内核的先例，本 PR 将类似的集成模式应用到了 DeepSeek-V4 模型，体现了在

ROCm 上利用 AITER 进行性能优化的演进脉络。