

PR #47984 完整报告

vllm-project/vllm

[ROCm][MiniMax-M3][Spec Decode] Support speculative decode with AITER sparse PA

合并时间: 2026-07-14 12:12

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/47984>

执行摘要

- 一句话: 支持 MiniMax-M3 投机解码的 ROCm 稀疏 PA 路径
- 推荐动作: 值得关注其设计模式: 通过重用预填充构建器扩展支持投机解码, 展示了以最小改动实现功能扩展的范例。对 vLLM 内核开发人员有参考价值, 尤其是稀疏注意力后端和投机解码的结合。

功能与动机

上游 PR #47287 添加了 AITER 稀疏 PA 路径, 但不支持投机解码, 阻碍了 MiniMax-M3 使用 EAGLE3/MTP 的稀疏 PA 快速路径。本 PR 以最小改动填补这一缺口。

实现拆解

1. 新增投机解码块表构造函数: 在 `vllm/models/minimax_m3/amd/ops/sparse_pa.py` 中新增 `minimax_m3_build_sparse_block_table_decode`。该函数将扁平化的 token 索引映射回请求 ID 和本地查询位置, 然后重用已有的预填充构建器 `minimax_m3_build_sparse_block_table_prefill`。当 `decode_query_len=1` 时直接回退到原有函数。
2. 移除多 token 解码限制并传递参数: 在 `vllm/models/minimax_m3/amd/sparse_attention_msa.py` 的 `forward` 方法中, 删除了对 `d.decode_query_len != 1` 的检查, 并将该参数传递给 `minimax_m3_sparse_attn_decode_aiter`。
3. 加固 Triton 内核边界条件: 在 `_build_sparse_block_table_kernel` 和 `_build_sparse_block_table_prefill_kernel` 中调整了 `valid` 掩码和 `causal_len` 计算, 加入 `has_tokens` 标志, 确保空解码行能正确产生空稀疏注意力。
4. 添加针对性测试: 在 `tests/kernels/attention/test_minimax_m3.py` 中添加两个 ROCm 专用测试: `test_aiter_sparse_block_table_handles_padded_decode_rows` 和 `test_aiter_decode_sparse_block_table_supports_spec_decode`。

关键文件:

- `tests/kernels/attention/test_minimax_m3.py` (模块 测试; 类别 `test`; 类型 `test-coverage`; 符号 `test_aiter_sparse_block_table_handles_padded_decode_rows`, `test_aiter_decode_sparse_block_table_supports_spec_decode`): 新增两个 ROCm 专用测试, 覆盖投机解码稀疏块表构建和填充行处理, 确保正确性。
- `vllm/models/minimax_m3/amd/ops/sparse_pa.py` (模块 稀疏 PA; 类别 `source`; 类型 `infrastructure`; 符号 `minimax_m3_build_sparse_block_table_decode`): 核心实现: 新增

投机解码专用稀疏块表构建函数，并加固 Triton 内核边界条件。

- `vllm/models/minimax_m3/amd/sparse_attention_msa.py`（模块 注意力前向；类别 `source`；类型 `data-contract`）：移除对多 token 解码的限制并传递 `decode_query_len` 参数，解锁投机解码路径。

关键符号：`minimax_m3_build_sparse_block_table_decode`,
`test_aiter_sparse_block_table_handles_padded_decode_rows`,
`test_aiter_decode_sparse_block_table_supports_spec_decode`

关键源码片段

`tests/kernels/attention/test_minimax_m3.py`

新增两个 ROCm 专用测试，覆盖投机解码稀疏块表构建和填充行处理，确保正确性。

```
@pytest.mark.skipif(
    not current_platform.is_rocm(),
    reason='MiniMax-M3 AITER sparse PA is ROCm-only',
)

def test_aiter_decode_sparse_block_table_supports_spec_decode():
    '''验证投机解码场景下稀疏块表构建的正确性。

    解码索引器将投机行展平为 req_id * decode_query_len + local_q,
    此测试验证 ROCm AITER 块表适配器使用相同的映射。
    '''

    from vllm.models.minimax_m3.amd.ops.sparse_pa import (
        PAGES_PER_SPARSE_BLOCK,
        minimax_m3_build_sparse_block_table_decode,
    )

    decode_query_len = 4
    # 三个请求：两个有效 (seq_len 260, 132)，一个填充 (0)
    seq_lens = torch.tensor([260, 132, 0], device='cuda', dtype=torch.int32)
    block_table = torch.tensor(
        [[5, 7, 11], [13, 17, 19], [0, 0, 0]],
        device='cuda', dtype=torch.int32,
    )

    # topk_idx 形状：[1, batch * decode_query_len, topk]
    topk_idx = torch.tensor(
        [[[2, 0, 3, -1], [0, 2, 3, -1], [2, 1, 0, -1], [1, 2, 0, -1],
          [1, 0, 2, -1], [0, 1, 2, -1], [1, -1, 0, -1], [0, 2, 1, -1],
          [0, 1, -1, -1], [0, 1, -1, -1], [0, 1, -1, -1], [0, 1, -1, -1]]],
        device='cuda', dtype=torch.int32,
    )

    # 调用被测试函数
    sparse_bt, sparse_ctx = minimax_m3_build_sparse_block_table_decode(
        topk_idx, block_table, seq_lens, decode_query_len
    )
```

```

torch.accelerator.synchronize()

# 手动计算期望值
expected_bt = torch.zeros_like(sparse_bt)
expected_ctx = torch.empty_like(sparse_ctx)
topk_cpu = topk_idx.cpu()[0].tolist()
seq_lens_cpu = seq_lens.cpu().tolist()

for row, selected_blocks in enumerate(topk_cpu):
    req_id = row // decode_query_len
    local_q = row - req_id * decode_query_len
    # 绝对查询位置 = seq_len[req_id] - decode_query_len + local_q
    query_pos = seq_lens_cpu[req_id] - decode_query_len + local_q
    causal_len = max(query_pos + 1, 0) # 填充行结果为 0
    if causal_len == 0:
        expected_ctx[row] = 0
        continue
    # 此处省略具体块表填充逻辑，详情见源代码
    # ...
assert torch.equal(sparse_ctx, expected_ctx)
assert torch.equal(sparse_bt, expected_bt)

```

vllm/models/minimax_m3/amd/ops/sparse_pa.py

核心实现：新增投机解码专用稀疏块表构建函数，并加固 Triton 内核边界条件。

```

@torch.no_grad()
def minimax_m3_build_sparse_block_table_decode(
    topk_idx: torch.Tensor, # [1, batch * decode_query_len, topk]
    block_table: torch.Tensor, # [batch, max_blocks]
    seq_lens: torch.Tensor, # [batch]
    decode_query_len: int,
) -> tuple[torch.Tensor, torch.Tensor]:
    '''为每个解码查询 token 构建一个紧凑的 page-16 稀疏块表行。

    当 decode_query_len == 1 时，直接回退到现有函数（非投机路径）。
    否则，将扁平化的 token 索引映射回请求 ID 和本地查询位置，
    然后重用预填充构建器生成表行。
    ...

    if decode_query_len == 1:
        # 单 token 解码：直接使用已有的块表构建器，保持不变
        return minimax_m3_build_sparse_block_table(topk_idx, block_table, seq_lens)

    total_q = topk_idx.shape[1]
    expected_q = block_table.shape[0] * decode_query_len
    # 断言：topk 行数必须等于 batch * decode_query_len
    assert total_q == expected_q, (
        f'MiniMax-M3 decode top-k rows must equal batch * decode_query_len: '
        f'{total_q} != {expected_q}'
    )

```

```

# 计算每个 token 在批中的位置索引
pos = torch.arange(total_q, dtype=torch.int32, device=topk_idx.device)
# 请求 ID = pos // decode_query_len
query_req_id = torch.div(pos, decode_query_len, rounding_mode='floor')
# 本地查询偏移 = pos % decode_query_len
q_offset = pos - query_req_id * decode_query_len
# 绝对查询位置 = seq_len[req_id] - decode_query_len + local_q
query_abs_pos = (
    seq_lens[query_req_id] - decode_query_len + q_offset
).to(torch.int32)

# 重用预填充构建器，传入请求 ID 和绝对位置
return minimax_m3_build_sparse_block_table_prefill(
    topk_idx, block_table, query_req_id, query_abs_pos
)

```

评论区精华

- 审查者 tjtanana 批准 PR，并提供了手动验证命令确认非投机解码路径不受影响。
- 作者提供了多次性能验证数据，确认约 8% 的吞吐提升和准确率改善；同时响应审查要求，提供了不同投机 token 数下的 draft acceptance 结果。
- Mergify 机器人指出合并冲突，作者通过 rebase 解决。
- 性能验证与结果 (performance): 性能提升被确认，PR 获得批准。
- 投机 token 数与 draft acceptance (correctness): 验证了不同配置下的正确性。
- 合并冲突处理 (other): 冲突已解决。

风险与影响

- 风险：改动集中在 ROCm AITER 稀疏 PA 路径，其他平台和注意力后端不受影响。Triton 内核边界加固是 MiniMax-M3 专用的，不涉及通用内核，风险低。非投机解码路径保持不变，性能回归风险小。测试覆盖了填充行和投机解码场景，降低了逻辑错误风险。主要风险是如果未来其他模型也使用相同的内核，边界条件变化可能引入问题，但目前没有。
- 影响：影响范围仅限使用 ROCm 平台、MiniMax-M3 模型、AITER 稀疏 PA 且启用投机解码的用户。带来约 8% 的吞吐提升和 TTFT 降低。API 和配置无变化，非投机解码用户无影响。代码量小（182 行新增），易于维护。
- 风险标记：ROCm 平台限制，MiniMax-M3 专用，Triton 内核边界调整

关联脉络

- PR #47287 [ROCm][MiniMax-M3] Add AITER sparse paged attention path: 本 PR 依赖 #47287 引入的 AITER 稀疏 PA 基础路径，在其基础上扩展了投机解码支持。