

PR #56628 完整报告

vllm-project/vllm

[ROCm][DSV4.1][Perf] Stride the DSA decode candidate mask over the live context

合并时间: 2026-09-15 04:47

原文链接: <http://prhub.com.cn/vllm-project/vllm/pull/56628>

执行摘要

- 一句话: ROCm DSA decode 掩码改为按行 end 步进, 省去整宽写 padding
- 推荐动作: 值得精读。该 PR 是“针对特定硬件实测调优 + 严格正确性测试”的教科书式内核优化: ① 通过让成本跟随数据而非 workspace 来消除死写; ② 通过 grid 静态化兼容 FULL cudagraph; ③ 通过“只比较 [0, end) 区域”的契约测试 + 变异验证证明测试非空洞; ④ 主动说明测量局限并采用 per-kernel 归因。建议重点学习其测试设计 (test_leaves_columns_past_end_untouched 锁定性能来源、test_cudagraph_replay_tracks_changing_ends 锁定图重放语义) 以及“共享 kernel 不动、变体挂调用侧”的平台差异化策略。

功能与动机

关联 RFC issue #56506 指出 DeepSeek-V4.1-Flash 在 MI355X 上 decode step 存在大量固定开销, 其中索引器链路占 decode step 约 11.4%。PR body 明确指出:

`rocm_fp8_paged_mqa_logits` 把整个 $(batch_size * next_n, max_model_len)$ workspace 交给 `apply_candidate_mask`, 在 `--max-model-len 1048576` 下每行启动 1024 个 program, 而实际上下文只有前 ~128 列有数据; 掩码内核成本跟踪的是行数而非上下文长度, 96 行时每次调用写 402 MB 的 store 全是死工作。PR 的目标是让成本跟随 live context 而非 workspace 宽度, 同时保持 FULL cudagraph 捕获有效性。

实现拆解

1. 新增 strided 掩码内核与包装函数: 在 `vllm/v1/attention/ops/rocm_aiter_mla_sparse.py` 中新增 `_mask_candidates_strided_kernel` (Triton JIT) 与 `_apply_candidate_mask_strided`。内核启动二维 grid (rows, $\min(128, \text{cddiv}(\text{width}, 1024))$), 每个 program 以 $\text{step} = \text{num_programs}(1) * \text{TILE}$ 沿列轴 stride, while $\text{tile_start} < \text{end}$ 循环只处理到每行 end 所在 tile 边界; `grid_cols` 仍由 `width` 推导, 因此 FULL cudagraph 捕获后重放时 grid 静态不变, 仅循环 trip count 随 device memory 中的 end 变化。
2. 保持共享 kernel 不变, 变体只挂在 ROCm 调用侧: PR 早期版本直接改 `model_executor/kernels/attention/dsa/candidate_blocks.py` 的 `apply_candidate_mask`, 但 128 这个 grid 尺寸是针对 gfx950 的 256 CU 实测的调优常数, 作者无法在 NVIDIA 上测量权衡, 因此最终把变体放在唯一调用方 `rocm_aiter_mla_sparse.py` 中, 共享 kernel 未被改动, 避免向 CUDA 用户交付未经测量的 grid 变更。

3. 切换 decode 路径, prefill 路径保留共享 kernel: rocm_aiter_sparse_attn_indexer 中 paged decode 分支从 `_apply_candidate_mask` 切换到 `_apply_candidate_mask_strided` ; prefill chunks 传入的是 chunk 大小的 logits, 无优化空间, 继续走共享 kernel。新 `_apply_candidate_mask_strided` 内部复用 `_candidate_flags_kernel` 生成 flags, 在 flags 计算和掩码写入之间多分配一个 (rows, nblocks + 1) 的 flags 张量, 属于少量额外显存开销。
4. 正确性由测试锁定: 新增 tests/kernels/attention/test_rocm_aiter_candidate_mask.py (ROCm-gated), 覆盖 ragged ends (0/1/1023/1024/1025/width 边界)、非零 starts、row_repeat > 1、block_size 8/16、宽窄两种宽度、列 stride 2 等情形, 断言 [0, end) 区域与共享 kernel 逐 bit 一致 (rtol=0, atol=0, equal_nan=True), 并单独测试 test_leaves_columns_past_end_untouched 锁定 end 之后列不被写入、test_cudagraph_replay_tracks_changing_ends 验证捕获后对不同 end 长度重放正确。
5. 性能与精度验证: 独立 kernel benchmark 与 in-situ torch-profiler 双路测量, 并跑 gsm8k 5-shot 精度回归; 同时用两个 kernel 变异 (提前一 tile 停止、去掉 stride) 验证测试非空洞, 各使 9 个测试失败 8 个。

关键文件:

- vllm/v1/attention/ops/rocm_aiter_mla_sparse.py (模块 注意力内核; 类别 infra; 类型 infrastructure; 符号 `_mask_candidates_strided_kernel`, `_apply_candidate_mask_strided`): 核心变更文件: 新增 `_mask_candidates_strided_kernel` 与 `_apply_candidate_mask_strided`, 并把 rocm_aiter_sparse_attn_indexer 的 paged decode 分支切换到 strided 变体。这是性能收益的来源, 也承载了 grid 静态化与 end 边界语义的设计决策。
- tests/kernels/attention/test_rocm_aiter_candidate_mask.py (模块 注意力内核; 类别 test; 类型 test-coverage; 符号 `_inputs`, `_run_both`, `_assert_agrees_within_ends`, `test_matches_shared_kernel_ragged_ends`): 新增 ROCm-gated 单元测试, 是正确性把关的核心: 验证 strided kernel 与共享 kernel 在 [0, end) 区域逐 bit 一致, 并显式锁定 end 之后不写与 cudagraph 重放行为。作者还用两个 kernel 变异证明测试非空洞。

关键符号: `_mask_candidates_strided_kernel`, `_apply_candidate_mask_strided`, `_assert_agrees_within_ends`, `test_matches_shared_kernel_ragged_ends`, `test_leaves_columns_past_end_untouched`, `test_cudagraph_replay_tracks_changing_ends`

关键源码片段

vllm/v1/attention/ops/rocm_aiter_mla_sparse.py

核心变更文件: 新增 `_mask_candidates_strided_kernel` 与 `_apply_candidate_mask_strided`, 并把 rocm_aiter_sparse_attn_indexer 的 paged decode 分支切换到 strided 变体。这是性能收益的来源, 也承载了 grid 静态化与 end 边界语义的设计决策。

```
# 固定沿列轴 striding 的 program 数, 针对 gfx950 的 256 CUs 实测调优;
# 不是可移植的选择, 因此变体只放在 ROCm 调用侧, 不替换共享 kernel。
_MASK_GRID_COLS = 128
# 每个 program 每次迭代处理的列数; 工作停在每行 end 所在的 tile 边界,
```

不是 end 本身——多写一个 tile，但与共享 kernel 的掩码语义等价。

_MASK_TILE = 1024

```
@triton.jit(do_not_specialize=["width", "nblocks"])
def _mask_candidates_strided_kernel(
    logits, starts, ends, flags,
    stride_row, stride_col, stride_start, stride_end,
    width, nblocks, BLOCK_SIZE: tl.constexpr,
    HAS_STARTS: tl.constexpr, ROW_REPEAT: tl.constexpr,
    TILE: tl.constexpr,
):
    row = tl.program_id(0).to(tl.int64)
    start = tl.load(starts + row // ROW_REPEAT * stride_start) if HAS_STARTS else 0
    end = tl.load.ends + row // ROW_REPEAT * stride_end)
    edge = tl.load(flags + row * (nblocks + 1) + nblocks)
    # 固定步长: program 沿列轴互相错开，共同覆盖整行。
    step = tl.num_programs(1) * TILE
    tile_start = tl.program_id(1) * TILE
    # 共享 kernel 会 sanitize 到 end 之后的所有列；但没有消费者读那里：
    # top_k_per_row_decode 用同一份 seq_lens 推导 rowEnd，persistent_topk 与
    # cooperative_topk 也按该长度 clamp。停在 end 让成本跟随 live context
    # 而不是 max_model_len 的 workspace 宽度。
    while tile_start < end:
        cols = tile_start + tl.arange(0, TILE)
        valid = (cols >= start) & (cols < end) & (cols < width)
        block = (cols - start) // BLOCK_SIZE
        keep = tl.load(flags + row * (nblocks + 1) + block, valid, other=0)
        keep = (keep != 0) | ((cols == width - 1) & (edge != 0))
        # 只有保留块写回原值，其余写 -inf，与共享 kernel 的掩码规则一致。
        tl.store(
            logits + row * stride_row + cols * stride_col,
            -float("inf"),
            (cols < width) & ~(valid & keep),
        )
        tile_start += step

def _apply_candidate_mask_strided(
    logits: torch.Tensor,
    row_ks: torch.Tensor | None,
    row_ke: torch.Tensor,
    candidate_blocks: torch.Tensor,
    block_size: int,
    row_repeat: int = 1,
) -> None:
    """ROCM decode 版 apply_candidate_mask: 语义在 [0, end) 一致。"""
    rows, width = logits.shape
    if not rows or not width:
```

```

    return
nblocks = triton.cdiv(width, block_size)
# 先算候选块 flags, 和共享 kernel 使用同一 _candidate_flags_kernel。
flags = torch.empty((rows, nblocks + 1), device=logits.device, dtype=torch.uint8)
start_stride = row_ks.stride(0) if row_ks is not None else 0
_candidate_flags_kernel[(rows,),](candidate_blocks, row_ks, flags,
    *candidate_blocks.stride(), start_stride, width, nblocks,
    block_size, candidate_blocks.shape[1], row_ks is not None, row_repeat)
# grid_cols 由 width (张量形状) 推导, 所以 FULL cudagraph 捕获时 grid
# 是静态的, 重放有效; 只有循环 trip count 数据相关, 由设备内存决定。
# min() 保证窄宽度不会启动只会空转的 program。
grid_cols = min(_MASK_GRID_COLS, triton.cdiv(width, _MASK_TILE))
_mask_candidates_strided_kernel[(rows, grid_cols)](
    logits, row_ks, row_ke, flags,
    *logits.stride(), start_stride, row_ke.stride(0),
    width, nblocks, block_size, row_ks is not None, row_repeat, _MASK_TILE,
)

```

tests/kernels/attention/test_rocm_aiter_candidate_mask.py

新增 ROCm-gated 单元测试, 是正确性把关的核心: 验证 strided kernel 与共享 kernel 在 [0, end) 区域逐 bit 一致, 并显式锁定 end 之后不写与 cudagraph 重放行为。作者还用两个 kernel 变异证明测试非空洞。

```

# 只比较消费者真正能读的区域 [0, end), 而不是整行。
# 这是 strided kernel 与共享 kernel 的显式契约: strided 版故意不写
# end 之后的列, 所以全行比较会让测试永远失败, 而契约测试让性能优化
# 有了正确性边界。
def _assert_agrees_within_ends(ref, got, row_ke, row_repeat):
    for row in range(ref.shape[0]):
        end = int(row_ke[row // row_repeat])
        if end <= 0:
            continue
        torch.testing.assert_close(
            got[row, :end], ref[row, :end],
            rtol=0, atol=0, equal_nan=True,
            msg=lambda m, r=row, e=end: f"row {r} differs within [0, {e}): {m}",
        )

```

```

# 覆盖宽窄两种宽度: 上面 128 个 tile (128 * 1024) 时 grid 饱和、program
# 真正开始 stride, 下面则触发 min() 截断, 两个状态都要覆盖。
NARROW = 8192
WIDE = 262144

```

```

@pytest.mark.parametrize("width", [NARROW, WIDE])
@pytest.mark.parametrize("col_stride", [1, 2])
def test_matches_shared_kernel_ragged_ends(width, col_stride):
    # 覆盖 0、1、tile 边界 1023/1024/1025、以及 end 等于 width 的整行情形。

```

```

ends = [0, 1, 1023, 1024, 1025, width - 1, width, width // 2]
logits, row_ks, row_ke, candidates = _inputs(
    len(ends), width, ends, None,
    block_size=BLOCK_SIZE, col_stride=col_stride, dtype=torch.int32,
)
ref, got = _run_both(logits, row_ks, row_ke, candidates, BLOCK_SIZE, 1)
_assert_agrees_within_ends(ref, got, row_ke, 1)

```

锁定性能来源：end 之后必须保持 sentinel 原值，若未来有人“修”成
恢复全宽 sanitize，这个测试会失败而不是悄悄变慢。

```

def test_leaves_columns_past_end_untouched():
    ends = [0, 1, 1025, WIDE // 2]
    logits, row_ks, row_ke, candidates = _inputs(
        len(ends), WIDE, ends, None,
        block_size=BLOCK_SIZE, col_stride=1, dtype=torch.int32,
    )
    sentinel = 1.5
    logits.fill_(sentinel)
    _apply_candidate_mask_strided(logits, row_ks, row_ke, candidates, BLOCK_SIZE, 1)
    for row, end in enumerate(ends):
        # 允许写到 end 所在 tile 边界，这是 overshoot 的上限。
        touched = min(-(-end // _MASK_TILE) * _MASK_TILE, WIDE)
        tail = logits[row, touched:]
        assert torch.equal(tail, torch.full_like(tail, sentinel)), \
            f"row {row} (end {end}) was written past column {touched}"
        # 边界必须紧：end 很小时不能付出整行代价，这正是内核的意义。
        assert touched <= end + _MASK_TILE

```

cudagraph 语义：捕获会固化 grid，如果 program 数数据相关就会冻结捕获时
的长度。这里验证重放更长 / 更短的上下文仍与共享 kernel 在 [0, end) 一致。

```

def test_cudagraph_replay_tracks_changing_ends():
    rows = 4
    captured_ends = [1024, 4096, 8192, 2048]
    logits, row_ks, row_ke, candidates = _inputs(
        rows, WIDE, captured_ends, None,
        block_size=BLOCK_SIZE, col_stride=1, dtype=torch.int32,
    )
    base = logits.clone()
    # 捕获前先 warm up: 避免 Triton 首次编译和 autotune 进入图内。
    _apply_candidate_mask_strided(
        logits.clone(), row_ks, row_ke, candidates, BLOCK_SIZE, 1)
    torch.accelerator.synchronize()
    graph = torch.cuda.CUDAGraph()
    with torch.cuda.graph(graph):
        _apply_candidate_mask_strided(
            logits, row_ks, row_ke, candidates, BLOCK_SIZE, 1)
    for replay_ends in ([1, 131072, WIDE, 0], [WIDE, 1025, 3, 65536]):

```

```
row_ke.copy_(torch.tensor(replay_ends, device="cuda", dtype=row_ke.dtype))
logits.copy_(base)
graph.replay()
torch.accelerator.synchronize()
ref = base.clone()
apply_candidate_mask(ref, row_ks, row_ke, candidates, BLOCK_SIZE, 1)
_assert_agrees_within_ends(ref, logits, row_ke, 1)
```

评论区精华

该 PR 的 review 评论极少：[claude\[bot\]](#) 因是 fork PR 自动停用评审，[tjtanaa](#) 直接 APPROVED 并给出“LGTM”。真正的设计讨论主要体现在 PR body 和提交说明中，作者明确阐述了两个关键决策：其一，为何不直接改共享 kernel——[128](#) 是针对 gfx950 256 CUs 实测的调优常数，作者无法在 NVIDIA 硬件上测量，故将变体放在 ROCm 调用侧，避免向 CUDA 路径交付未经测量的 grid 变更；其二，为何只在 decode 路径切换——prefill chunks 传入 chunk 大小 logits，无收益空间。此外作者主动说明了测量局限：低并发下 end-to-end wall time 被 [cross_device_reduce](#) 的方差淹没 (>2 ms/step)，因此以 per-kernel 归因作为可信度量。

- 为什么不在共享 kernel 上直接改 (design): 接受该设计：网格变更仅限 ROCm decode 路径，共享 kernel 与 CUDA 路径不受影响。
- 正确性契约：只保证 [0, end) 一致 (correctness): 测试以 [0, end) 区域逐 bit 一致作为契约，并额外用 sentinel 锁定 end 后不写，防止未来回归。
- 测量局限与可信度 (question): per-kernel 归因作为主要度量，低并发 wall time 不作为结论依据。

风险与影响

- 风险：
 1. 依赖消费者扫描边界约定：strided kernel 故意不写端后区域，正确性依赖 top_k_per_row_decode 使用同一 seq_lens 推导 rowEnd、persistent_topk 与 cooperative_topk 对长度 clamp 不越界读取。若未来消费者逻辑变化（如读整个 workspace 或改用不同 end 来源），会产生未定义数据读取，属于隐式跨模块契约。
 2. 平台特异性调优常数：_MASK_GRID_COLS = 128 是针对 gfx950 256 CUs 的实测值，未经 NVIDIA 测量；变体仅挂在 ROCm 路径，但若其他 ROCm 型号（如 gfx90a、gfx1200）CU 数差异较大，性能可能退化，尽管正确性不受影响。
 3. 额外 flags 张量分配：_apply_candidate_mask_strided 每次调用在 _candidate_flags_kernel 前分配 (rows, nblocks + 1) 的 flags 张量，与共享 kernel 相比多一次分配（共享 kernel 内部也可能类似），在 decode 高频调用下需关注显存碎片或分配开销。
 4. CUDAGraph 重放边界：grid 静态化依赖 width 在捕获与重放间不变；若未来出现 width 可变但 grid 仍静态的调用路径，需重新评估。当前测试已覆盖长短 end 重放，但未覆盖 width 变化。

5. 测试仅限 ROCm: 所有正确性测试都被 `skipif(not is_rocm())` 门控, CUDA 路径完全不受本 PR 影响, 但也意味着 strided 变体在非 ROCm 平台无回归保障。 - 影响: 影响范围集中在 ROCm 平台上 DeepSeek-V4.1-Flash (及复用 `rocm_aiter_sparse_attn_indexer` 的稀疏 MLA 模型) 的 paged decode 路径。受益场景是 `max_model_len` 远大于实际上下文的长上下文 / 高并发 decode: 96 行时掩码 kernel 从 0.1236 ms 降至 0.0201 ms (6.16x), decode step 索引器链路占比从 11.4% 降至 10.0%; 并发 1/4/16 时 step 级 kernel 时间分别加速 2.7x/4.3x/6.0x。对 prefill、CUDA、CPU/XPU 等路径零影响。团队层面, 该 PR 是 #56506 RFC 下 DSv4.1 ROCm 性能优化系列的一环, 为后续更多“按 live context 而非 workspace 计算”的 kernel 优化提供了可复用的测试模式与设计范式。 - 风险标记: 依赖隐式跨模块契约, 平台特定调优常数未经 CUDA 验证, 新增 flags 张量分配开销, 测试仅限 ROCm

关联脉络

- PR #56506 [RFC]: DeepSeek-V4.1-Flash performance on ROCm: 本 PR 直接声明为 Part of #56506, 是 RFC 中索引器 / 掩码链路优化的落地项。
- PR #56513 [ROCm][DSv4.1][Perf] Fold the mHC post step into the delayed pre projection: 同系列 ROCm DSv4.1 性能优化, 针对 mHC 链路, 与本 PR 共同降低 decode step 固定开销。
- PR #56633 [Perf][DSv4.1] Fold the mHC post block into the delayed pre projection: NVIDIA 侧对应优化, 说明 DSv4.1 性能系列按平台分别推进, 本 PR 采用“平台变体不污染共享 kernel”策略。
- PR #56503 [ROCm][DSv4.1] mHC pre AITER migration: RFC 中另一项关键优化 (mHC pre 迁移到 AITER), 与本文掩码优化同属 #56506 的依赖链, 影响同一批精度与性能基准数据。