PR #51647 完整报告

vllm-project/vllm

[ROCm] Pad non-aligned AITER MLA heads

合并时间：2026-08-19 05:37

原文链接：http://prhub.com.cn/vllm-project/vllm/pull/51647

执行摘要

- 一句话：AITER MLA 填充非对齐 head，Kimi-K3 TP4 吞吐提升约 40%
- 推荐动作：值得精读。该 PR 有三个值得借鉴的设计决策：一是在没有 capability API 时用“读 JIT 源码去空白搜索特征串 + lru_cache”做弱依赖探测，并在失败时保守回退；二是 reducer 与 metadata planner 必须双探测的完备性意识，避免只查一半导致 kernel 启动前崩溃；三是把填充语义（repeat_interleave vs tile-and-slice、零拷贝透传）收敛到单一 helper，并让元数据尺寸与 launch shape 同源，降低两处不同步的回归风险。

功能与动机

AITER 的 asm persistent decode 要求 head 数为 16 的倍数，而 Kimi-K3 在 TP4 下每 rank 恰好 24 heads，落在旧版 AiterMLAHelper 的不支持区间，只能回退到 Triton MLA。PR body 的基准显示 Triton MLA 在 C16/C24 两个 cohort 上 TPOT 达 62.7-76.1 ms；作者的目标是让 24 heads/rank 直接使用 AITER MLA（约 41.0-46.8 ms），并通过能力探测保证旧 AITER 版本仍可走填充路径正确运行。

实现拆解

以 AiterMLAHelper 的头部填充语义为中心，分四步落地。

1. 能力探测（vllm/v1/attention/backends/mla/rocm_aiter_mla.py）：新增 _aiter_mla_native_h24_reducer_supported、_aiter_mla_native_h24_metadata_supported、_aiter_mla_native_h24_supported 三个函数（带 functools.lru_cache）。由于 AITER 尚无公开 capability API，探测通过读取其 JIT 源码目录中的 reduce.cu 与 metadata/v1_2_device.cuh 文本、去空白后搜索 MLA_REDUCE_CASE_EF(NUM_HEAD,24,HEAD_DIM,512, 与 num_heads==24 特征串实现。关键设计：reducer 与 metadata planner 的形状分派相互独立，必须两者都命中才启用原生 H24，避免把 H24 送进不支持它的 planner。
2. 填充逻辑泛化（vllm/v1/attention/backends/mla/rocm_aiter_mla.py）：get_actual_mla_num_heads 从 max(num_heads, 16) 改为“H24 且原生支持则 24，否则向上取整到 16 倍数”；is_valid_num_heads 合法区间放宽到 1-128（超出后仍要求 16 倍数），并预留 _AITER_UNSUPPORTED_HEADS 黑名单常量。get_mla_padded_q / get_mla_unpadded_o 保留 16 的因子（1/2/4/8）的 repeat_interleave / strided 撤销路径，对 24 -> 32 这类非整除关系改用 tile-and-slice：q.repeat(1, reps, 1)[:, :m, :].contiguous()，输出只取前 num_heads。因为 MLA 各 query head 共享 KV、彼此独立

，padding head 不会污染真实 head。

3. 元数据尺寸同步（rocm_aiter_mla.py 与 rocm_aiter_mla_sparse.py）：两处 _num_attention_heads 初始化从 max(16, self.num_heads) 改为 AiterMLAHelper.get_actual_mla_num_heads(self.num_heads)，确保 get_mla_metadata_info_v1 分配的 persistent work-split / reduce 缓冲与传给 mla_decode_fwd（及 sparse decode kernel）的实际 padded shape 一致；否则 24 -> 32 的 launch 会越界或产生错误的 split/reduce 调度。
4. 测试配套：tests/kernels/attention/test_rocm_aiter_mla_head_padding.py 用 autouse fixture _disable_native_h24 默认强制走填充路径，新增 H24 -> H32 填充与裁剪、reducer 支持但 metadata 不支持时仍填充、双满足时零拷贝透传、17/24/31 经 H32 往返、H32 零拷贝等用例；tests/v1/attention/test_rocm_aiter_mla_mtp_split.py 把 head 扫描扩到 24，断言元数据尺寸走 get_actual_mla_num_heads，并在 persistent metadata gate 参数化中加入 (1, 1, 24, "auto", True)。

关键文件：

- vllm/v1/attention/backends/mla/rocm_aiter_mla.py（模块 注意力后端；类别 source；类型 core-logic；符号 _aiter_mla_native_h24_reducer_supported, _aiter_mla_native_h24_metadata_supported, _aiter_mla_native_h24_supported, AiterMLAHelper.get_actual_mla_num_heads）：核心实现文件：新增 AITER 原生 H24 能力探测、泛化头部填充语义、统一元数据尺寸，直接决定 Kimi-K3 TP4 的 kernel 选择与填充正确性。
- tests/kernels/attention/test_rocm_aiter_mla_head_padding.py（模块 头数填充；类别 test；类型 test-coverage；符号 _disable_native_h24, test_h24_query_is_tile_padded_to_h32, test_h24_output_discards_h32_padding_heads, test_h24_reducer_without_metadata_still_pads_to_h32）：覆盖 H24 -> H32 填充 / 裁剪、原生 H24 双条件门控、17/24/31 往返与 H32 零拷贝，是填充语义正确性的主要保障。
- vllm/v1/attention/backends/mla/rocm_aiter_mla_sparse.py（模块 稀疏注意力；类别 source；类型 core-logic；符号 AiterMLAHelper.get_actual_mla_num_heads）：sparse 后端同样参与 head 填充：元数据尺寸从 max(16, ...) 改为 get_actual_mla_num_heads，避免 padded launch 越界。
- tests/v1/attention/test_rocm_aiter_mla_mtp_split.py（模块 投机切分；类别 test；类型 test-coverage；符号 AiterMLAHelper.get_actual_mla_num_heads, test_mtp_builder_init_sizes_native_fp8_metadata, test_persistent_metadata_gate）：校验 MTP 与 persistent metadata gate 在 head 填充语义下仍正确，head 扫描扩到 24。

关键符号：_aiter_mla_native_h24_reducer_supported, _aiter_mla_native_h24_metadata_supported, _aiter_mla_native_h24_supported, AiterMLAHelper.get_actual_mla_num_heads, AiterMLAHelper.get_mla_padded_q, AiterMLAHelper.get_mla_unpadded_o, AiterMLAHelper.is_valid_num_heads, AiterMLAHelper.check_num_heads_validity

关键源码片段

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

核心实现文件：新增 AITER 原生 H24 能力探测、泛化头部填充语义、统一元数据尺寸，直接决定 Kimi-K3 TP4 的 kernel 选择与填充正确性。

```
# AITER 的 asm persistent decode 要求 head 数为 16 的倍数。以下两个探测函数
# 读取 AITER JIT 源码判断是否原生支持 H24/512 形状。AITER 尚没有公开的
# capability API，只能以源码中是否存在对应分支作为依据。
@functools.lru_cache(maxsize=1)
def _aiter_mla_native_h24_reducer_supported() -> bool:
    """AITER 的 JIT reducer 是否支持原生 H24/512 形状。"""
    try:
        from aiter.jit.core import AITER_CSRC_DIR

        reduce_source = Path(AITER_CSRC_DIR) / "kernels" / "mla" / "reduce.cu"
        source = "".join(reduce_source.read_text(encoding="utf-8").split())
    except (ImportError, OSError):
        return False
    return "MLA_REDUCE_CASE_EF(NUM_HEAD,24,HEAD_DIM,512," in source


@functools.lru_cache(maxsize=1)
def _aiter_mla_native_h24_metadata_supported() -> bool:
    """AITER 的快速 MLA metadata planner 是否接受原生 H24。

    reducer 与 metadata planner 的形状分派相互独立：只探测 reducer 可能把
    H24 路由进一个在 attention kernel 启动前就拒绝它的 planner。
    """
    try:
        from aiter.jit.core import AITER_CSRC_DIR

        metadata_source = (
            Path(AITER_CSRC_DIR) / "kernels" / "mla" / "metadata" / "v1_2_device.cuh"
        )
        source = "".join(metadata_source.read_text(encoding="utf-8").split())
    except (ImportError, OSError):
        return False
    return "num_heads==24" in source


class AiterMLAHelper:
    _AITER_MIN_MLA_HEADS: Final = 16
    # 允许填充的最大 head 数；超过后只有 16 的倍数合法。
    _AITER_MAX_PADDED_MLA_HEADS: Final = 128

    @staticmethod
    def get_actual_mla_num_heads(num_heads: int) -> int:
        # 原生 H24 直接透传，避免 24 -> 32 的填充开销。
        if num_heads == 24 and _aiter_mla_native_h24_supported():
```

```
            return num_heads
        m = AiterMLAHelper._AITER_MIN_MLA_HEADS
        # 其余非对齐 head 数向上取整到下一个 16 倍数（如 24 -> 32）。
        return -(-num_heads // m) * m

    @staticmethod
    def get_mla_padded_q(num_heads: int, q: torch.Tensor) -> torch.Tensor:
        m = AiterMLAHelper.get_actual_mla_num_heads(num_heads)
        if num_heads == m:
            return q # 原生或已对齐，零拷贝透传。
        if m % num_heads == 0:
            # 1/2/4/8 等 16 的因子沿用 repeat_interleave 老路径。
            return q.repeat_interleave(m // num_heads, dim=1)
        # 非整除关系（如 24 -> 32）用 tile-and-slice：把 query head 平铺后
        # 截到恰好 m 个。MLA 各 query head 共享 KV、彼此独立，填充 head 不会
        # 影响 [0:num_heads]，后续从输出里裁掉即可。
        reps = -(-m // num_heads) # ceil(m / num_heads)
        # 平铺再切片会产生非连续视图，而 asm decode 按 [tokens, m, head_dim]
        # 紧凑读取，因此这里物化一份连续拷贝。
        return q.repeat(1, reps, 1)[:, :m, :].contiguous()

    @staticmethod
    def get_mla_unpadded_o(num_heads: int, o: torch.Tensor) -> torch.Tensor:
        m = AiterMLAHelper.get_actual_mla_num_heads(num_heads)
        if num_heads == m:
            return o
        if m % num_heads == 0:
            return o[:, :: m // num_heads, :]
        # 撤销 tile-padding：真实 head 就是前 num_heads 个。
        return o[:, :num_heads, :]
```

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

sparse 后端同样参与 head 填充：元数据尺寸从 max(16, ...) 改为 get_actual_mla_num_heads，避免 padded launch 越界。

```
# ----- Persistent MLA metadata buffers -----
# 关键配套改动：稀疏解码的 _num_attention_heads 不再用 max(16, num_heads),
# 而是与 dense 路径统一走 AiterMLAHelper.get_actual_mla_num_heads,
# 保证 get_mla_metadata_info_v1 分配的持久化 work-split / reduce 缓冲
# 与传入 sparse decode kernel 的实际 padded shape（如 24 -> 32）一致。
from aiter import dtypes, get_mla_metadata_info_v1

# Keep metadata sizing consistent with the padded tensor shape passed
# to the sparse decode kernel.
self._num_attention_heads = AiterMLAHelper.get_actual_mla_num_heads(
    self.num_heads
)
```

评论区精华

评审与流程讨论集中在三件事。一是原生 H24 能力检查的完备性：作者邀请评审时说明“if AITER supports native H24, vLLM uses 24 heads directly without padding; otherwise, it falls back to padding H24 to H32”，而 commit Require metadata support for native H24 MLA 印证了讨论结论——早期版本只探测 reducer，评审后补上 metadata planner 探测，代码注释明确写道两者 shape dispatch 相互独立，只查一半会把 H24 路由进拒绝它的 planner。二是 pre-commit 校验反复失败：mergify bot 至少四次提示，作者分别提交 “Apply pre-commit formatting” 与 “Apply ruff formatting” 两个格式化 commit 才通过。三是流程层面：fork PR 的 /ci run 需要 write 权限，作者多次请求 tjtanaa 与 AndreasKaratzas 触发；AndreasKaratzas 添加 ready label 以运行完整 ROCm 套件，claude[bot] 自动评审对 fork 关闭，最终由 tjtanaa 人工 approve。

- 原生 H24 能力探测是否只查 reducer 即可 (correctness): 原生 H24 需要 reducer 与 metadata planner 双探测，两者同时满足才透传 24 heads；否则回退到 24 -> 32 填充。
- pre-commit 与 ruff 格式化反复失败 (style): 最终 head d894f2f 通过 CI 校验，格式化提交合入。
- fork PR 的 CI 触发与人工评审协调 (other): tjtanaa 触发 CI 并 approve，PR 合入 main。

风险与影响

- 风险：技术风险集中在四处。第一，能力探测依赖 AITER JIT 源码路径与特征串（aiter.jit.core.AITER_CSRC_DIR、reduce.cu、v1_2_device.cuh），AITER 升级若改变文件布局、宏写法或换行，探测会失效；失败时保守回退到填充路径，安全但丢失原生 H24 收益。第二，get_mla_padded_q 的 tile-and-slice 需要 contiguous() 物化，decode 热路径增加一份 [tokens, 32, head_dim] 拷贝与约 1/3 冗余头部注意力计算，实测收益仍显著，但该开销值得后续跟踪。第三，head 数合法区间从“<16 或 16 倍数”放宽到“1-128 或 16 倍数”，17-128 之间的非对齐 head 数首次进入 asm 填充路径，若某些数值触发 AITER kernel 内部隐藏限制（如 head_dim、block 上限），会引入新失败模式，_AITER_UNSUPPORTED_HEADS 是预置的逃生舱。第四，元数据尺寸一致性是正确性关键，dense 与 sparse 两处 _num_attention_heads 已收敛到 get_actual_mla_num_heads，若未来其它调用点（如 MTP verify 路径）未同步，可能产生越界写或错误 split/reduce 调度。兼容性方面，改动集中在 ROCm AITER MLA 后端，NVIDIA/ 其他平台不受影响，老 AITER 无 H24 分支时自动走填充，向后兼容。
- 影响：部署侧，Kimi-K3 在 ROCm MI355X 的 TP4 配置（24 heads/rank）从 Triton MLA 切到 AITER MLA，100K token 长上下文解码 TPOT 下降 34.6%-38.5%、吞吐提升 39.5%-45.9%，GSM8K-100 为 99/100、无 malformed 输出。代码侧，AiterMLAHelper 成为 dense 与 sparse 两条 MLA 路径共同的头部语义中心，能力探测与填充逻辑集中在 backend 层，测试体系新增对“填充往返”“原生 H24 门控”“零拷贝透传”的确定性覆盖。影响范围限定在 v1 attention 后端的 ROCm + AITER 组合，对未启用 AITER MLA 的配置零影响。
- 风险标记：核心解码路径变更，依赖 AITER 内部源码探测，head 数合法性区间放宽，decode 热路径引入拷贝

关联脉络

- PR #52046 [nv] add pcp support in dsv3.2: 同属 ROCm AITER MLA 稀疏路径（vllm/v1/attention/ops/rocm_aiter_mla_sparse.py），本 PR 对 sparse 后端元数据尺寸做了配套修改，两者共同演进 AITER sparse MLA 的可用性。
- PR #52293 [ROCm][Perf] Enable fused KDA decode on gfx942 (MI325X): 同为 Kimi-K3 在 ROCm 上的解码性能解锁（gfx942 fused KDA decode），与本 PR 一起构成 Kimi-K3 ROCm 解码加速主线。
- PR #52381 Harden DeepSeek V3.2 fused kernel grids: DSV3.2 fused kernel grid 越界加固，与本 PR 同属对 kernel 形状合法性与边界条件的防护主题。
- PR #52512 [Bugfix][MLA] Do not use Dense MHA for GLM-5.2: MLA 注意力路径的 head 路由决策（dense MHA vs 原生 MLA），与本 PR 的 head 数合法性与 kernel 选择逻辑相关。