# PR #36852 完整报告

sgl-project/sglang

[ROCm][Bugfix] Use token-level KV indices in the aiter ASM context-prefill gather

合并时间：2026-08-29 08:17

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/36852

## 执行摘要

- 一句话：修复 aiter ASM prefill 在 page_size>1 时 KV 索引错误
- 推荐动作：值得精读，尤其是对 AMD 注意力内核与 KV 元数据口径感兴趣的同学。核心收益有三点：一是识别出 kv_indptr/kv_indices 的 token 级语义并简化 gather；二是把不可测试的内联逻辑提取为模块级纯函数并补回退日志；三是用随机打乱页面的构造暴露 page_size=1 掩盖边界缺陷的测试设计。

## 功能与动机

36758 为 gfx950 的 fp8 上下文块 prefill 增加了 ASM varlen 路径，并在默认 --page-size 1 下测得 +4-10% 吞吐提升。但该路径把 kv_indptr/kv_indices 当作页粒度，而实际上 AiterIndicesUpdaterPrefill 生成 kv_indptr = cumsum(seq_lens)，create_flashinfer_kv_indices_triton 为每个 token 写一条 kv_indices，token t 的池槽位就是 kv_indices[kv_indptr[i] + t]，无需 page 换算。page_size>1 时旧公式要么越界回退、要么静默读错槽位，PR body 给出 gsm8k 准确率从 0.995 跌到 0.53/0.26 的严重回退；page_size=1 下两种口径数字相同，缺陷被掩盖。

## 实现拆解

1. 修正索引口径（python/sglang/srt/layers/attention/aiter_backend.py）：删除 page = self.page_size 参与的旧公式（page_slot = kv_indptr[seq_ids] + pos_in_seq // page、tok_idx = kv_pages[page_slot] * page + pos_in_seq % page），改为 token 级 kv_slot = kv_indptr[seq_ids] + pos_in_seq、tok_idx = kv_indices[kv_slot]，self.page_size 不再出现在 gather 中。
2. 提取模块级纯函数：从 forward_extend 中抽出 _asm_context_prefill_gather_indices(kv_indptr, kv_indices, seq_lens, num_kv_slots, forward_mode)，纯张量运算可在 CPU 上直接验证；调用点通过返回 None 表示元数据不一致并回退到 ck_tile，返回 (tok_idx, cu_k)

供 ASM 内核使用。

3. 规范回退日志：第二道边界检查（tok_idx 越出池）原先静默回退，现与第一道一样打印 forward_mode、bs、越界值与表长，便于线上判断 ASM 路径是否真的命中，还是悄悄转换成 ck_tile。
4. 新增 CPU 单测（test/registered/attention/test_aiter_asm_prefill_gather.py）：用 _paged_pool 随机打乱页面布局模拟真实分配器，覆盖 page_size 1/16/64、多序列、未对齐 seq_lens、单序列以及元数据过短 / 池越界回退分支，直接校验 tok_idx 与 cu_k。测试注册到 stage-b-test-1-gpu-small-amd，仅需 CPU，成本很低。

关键文件：

- python/sglang/srt/layers/attention/aiter_backend.py（模块 注意力后端；类别 source；类型 core-logic；符号 _asm_context_prefill_gather_indices）：核心修改文件：修正 gfx950 ASM 上下文块 prefill 的 KV gather 索引口径，从页粒度改为 token 级，并提取模块级纯函数、补齐回退日志。
- test/registered/attention/test_aiter_asm_prefill_gather.py（模块 索引测试；类别 test；类型 test-coverage；符号 _paged_pool, _token_level_metadata, TestAsmPrefillGatherIndices, _check）：新增 CPU 单测，用随机打乱页面的池验证 page_size 1/16/64 下 gather 索引正确与回退分支，是回归防护的关键。

关键符号：_asm_context_prefill_gather_indices, _paged_pool, _token_level_metadata, _check, test_page_sizes, test_single_sequence, test_unaligned_seq_lens, test_falls_back_when_metadata_is_short, test_falls_back_when_slot_exceeds_pool

## 关键源码片段

### python/sglang/srt/layers/attention/aiter_backend.py

核心修改文件：修正 gfx950 ASM 上下文块 prefill 的 KV gather 索引口径，从页粒度改为 token 级，并提取模块级纯函数、补齐回退日志。

```
def _asm_context_prefill_gather_indices(
    kv_indptr: torch.Tensor,
    kv_indices: torch.Tensor,
    seq_lens: torch.Tensor,
    num_kv_slots: int,
    forward_mode=None,
):
    """计算 ASM 上下文块 prefill 的 KV 池槽位映射。

    kv_indptr/kv_indices 在任意 page_size 下都是 token 级：
    AiterIndicesUpdaterPrefill 设置 kv_indptr = cumsum(seq_lens)，
    而 create_flashinfer_kv_indices_triton 为每个 token 写一条 kv_indices，
    所以序列 i 的 token t 位于 kv_indices[kv_indptr[i] + t]，
    这里不需要再做任何 page 算术。
    """
    # 把元数据统一成 long，并截断到 kv_indices 实际拥有的条目数，
    # 避免 mixed/spec 批次下 gather 越过表尾。
```

```
    bs = kv_indptr.numel() - 1
    device = kv_indices.device
    kv_indptr = kv_indptr.to(torch.long)
    seq_lens = seq_lens.to(device=device, dtype=torch.long)
    seq_lens = torch.minimum(seq_lens, kv_indptr[1:] - kv_indptr[:bs])

    # 构造 token 级索引：cu_k 是每个序列的累计起点，
    # seq_ids/pos_in_seq 把扁平位置还原成 (seq, pos) 对。
    total_k = int(seq_lens.sum().item())
    cu_k = torch.zeros(bs + 1, dtype=torch.long, device=device)
    torch.cumsum(seq_lens, 0, out=cu_k[1:])
    seq_ids = torch.repeat_interleave(torch.arange(bs, device=device), seq_lens)
    pos_in_seq = torch.arange(total_k, device=device) - cu_k[seq_ids]
    kv_slot = kv_indptr[seq_ids] + pos_in_seq

    # 第一道边界检查：kv_slot 必须落在 kv_indices 表内，
    # 否则说明元数据口径不一致，回退到 paged kernel。
    if total_k and int(kv_slot.max().item()) >= kv_indices.numel():
        logger.warning(
            "[asm-context-prefill] metadata mismatch, falling back:"
            " mode=%s bs=%s kv_slot_max=%s kv_indices=%s seq_lens=%s kv_indptr=%s",
            forward_mode,
            bs,
            int(kv_slot.max().item()),
            kv_indices.numel(),
            seq_lens.tolist(),
            kv_indptr.tolist(),
        )
        return None

    # 直接按 token 级表 gather 出池槽位，再做第二道边界检查：
    # tok_idx 必须落在 KV 池内，否则同样回退并通过日志暴露原因。
    tok_idx = kv_indices[kv_slot].to(torch.long)
    if total_k and int(tok_idx.max().item()) >= num_kv_slots:
        logger.warning(
            "[asm-context-prefill] gather index out of pool, falling back:"
            " mode=%s bs=%s tok_idx_max=%s num_kv_slots=%s",
            forward_mode,
            bs,
            int(tok_idx.max().item()),
            num_kv_slots,
        )
        return None

    # 返回 (token 到池槽的映射，每个序列的累计 token 数)，供 ASM 内核使用。
    return tok_idx, cu_k
```

test/registered/attention/test_aiter_asm_prefill_gather.py

新增 CPU 单测，用随机打乱页面的池验证 page_size 1/16/64 下 gather 索引正确与回退分支，是回归防护的关键。

```
def _paged_pool(seq_lens, page_size, seed, headroom=2):
    """用随机打乱的页面把每条序列铺在池中，模拟 page allocator 行为。

    返回 (req_to_token, num_kv_slots)：req_to_token[i][t] 是序列 i 第 t 个
    token 所在的池槽位。页面随机发放，保证正确的 gather 不能依赖序列在池中
    连续，从而在单测里复现 page_size>1 才会触发的索引错误。
    """
    num_pages = headroom * sum((n + page_size - 1) // page_size for n in seq_lens)
    g = torch.Generator().manual_seed(seed)
    free_pages = torch.randperm(num_pages, generator=g).tolist()
    req_to_token = []
    for n in seq_lens:
        slots = []
        for _ in range((n + page_size - 1) // page_size):
            base = free_pages.pop() * page_size
            slots.extend(range(base, base + page_size))
        req_to_token.append(slots[:n])
    return req_to_token, num_pages * page_size


def _token_level_metadata(req_to_token):
    """按 AiterIndicesUpdaterPrefill 的方式构造 kv_indptr/kv_indices。"""
    seq_lens = torch.tensor([len(s) for s in req_to_token], dtype=torch.long)
    kv_indptr = torch.zeros(len(req_to_token) + 1, dtype=torch.long)
    torch.cumsum(seq_lens, 0, out=kv_indptr[1:])
    kv_indices = torch.tensor(
        [slot for slots in req_to_token for slot in slots], dtype=torch.int32
    )
    return kv_indptr, kv_indices, seq_lens
```

## 评论区精华

本 PR 没有实质性审查分歧：HaiShaw 直接批准，zijiecode 仅留 Thank you! LGTM。最有价值的讨论集中在 PR body 的根因分析：作者指出 kv_indptr/kv_indices 在每种 page_size 下都是 token 级，AiterIndicesUpdaterPrefill 用 cumsum(seq_lens) 生成 kv_indptr，create_flashinfer_kv_indices_triton 逐 token 写 kv_indices，因此旧代码中的 // page 与 % page 是双重换算；page_size=1 时两种口径数值完全相同，加上单序列 + 页对齐布局的测试无法暴露错误，导致该缺陷长期未被发现。

- 批准与合并 (other): 无实质分歧，已合并。

## 风险与影响

- 风险：主要风险集中在 AMD/gfx950 专用的 fp8 上下文块 prefill 路径：（1）若未来出现真正页粒度的 kv_indices 生产者，token 级公式会失效，但目前所有生产者均按 token 级写入，且文件内已有页粒度与 token 级两种口径并存，靠两道边界检查兜底，建议后续统一

元数据口径；（2）回退仍依赖边界检查，num_kv_slots 或 kv_indices 长度错误会改变是回退还是越界读的判断，新增测试覆盖了这两类场景；（3）forward_extend 调用点任何元数据不一致都会导致 ASM 路径静默退化为 ck_tile，虽然正确但丢掉性能，新增日志可辅助 排查；（4）本变更只影响 page_size > 1 的运行，默认 page_size=1 无行为变化，对现有 CUDA/ 默认部署无回归面。

- 影响：影响范围主要为使用 ROCm/gfx950（如 MI355X）与 fp8 KV cache、head_dim=256 并开启上下文块 prefill 的用户。修复后 page_size>1 不再出现静默错误结果，且能真正命中 ASM 优化路径；同时把索引逻辑变成可在 CPU 上单测的纯函数，降低后续内核改动风险。默认 --page-size 1 部署完全不变。团队侧新增一个 AMD CI 注册的小型单测（stage-b-test-1-gpu-small-amd），测试成本低。PR 页面显示的三个 CI 状态均为失败，合并时需确认是否与本次改动相关。
- 风险标记：AMD/gfx950 专用路径，回退依赖边界检查，核心注意力路径改动，默认 page_size=1 无行为变化

## 关联脉络

- PR #36758 ASM fp8 varlen context-chunk prefill for gfx950: 本 PR 直接修复 #36758 在 page_size>1 时的 KV 索引缺陷；PR body 明确引用该 PR 为动机来源。