

PR #39487 完整报告

sgl-project/sglang

[Fix][DCP] Localize widened KV ids in MLA retraction CPU backup/restore

合并时间: 2026-09-15 13:33

原文链接: <http://prhub.com.cn/sgl-project/sglang/pull/39487>

执行摘要

- 一句话: 修复 DCP 下 MLA 重删 CPU 备份 / 恢复的 KV 索引越界崩溃
- 推荐动作: 值得精读, 尤其对维护 DCP / PD 分离 / HiCache / KV 内存池的同学。重点关注三处设计决策: ① 把索引换算从 host pool 方法提升为 layers/dcp/layout.py 纯函数的动机与收益; ② `_write_loc_dcp_span + write_loc_is_dcp_resolved` 这对契约如何避免统一池上的双重换算; ③ 为什么可以安全删除「整加宽页」断言——理解 `seqlen - 1` 的 ragged 前缀与「起始页对齐」这个更弱前提的差别。若只关心结论, 可记住两点: 崩溃触发条件是 `DCP > 1` 时用加宽 id 直接索引 rank 局部 KV buffer; 修复方式是 `indices[dcp_rank::dcp_size] // dcp_size`。合并后建议跟进两件事: 为 `maybe_dcp_kernel_indices` 补一个起始对齐的显式校验或更强的测试守护, 以及排查作者提到的「恢复请求重复 token」独立缺陷。

功能与动机

Issue#38645 记录了线上的具体崩溃: 在 PD-disaggregation decode 节点、DCP8、Kimi-K3 (MLA + KDA 混合) 配置下, decode 容量压力触发 `retract_decode` 后, 调度器在 `Req.offload_kv_cache -> TokenToKVPoolAllocator.get_cpu_copy -> MLATokenToKVPool.get_cpu_copy` 的 `kv_buffer[layer_id][chunk_indices]` 处抛 CUDA device-side assert。Issue 明确给出的根因是索引空间不一致: 「req_to_token stores DCP-widened logical KV indices」而「per-rank MLA kv_buffer stores only rank-local physical rows」, 正常 MLA 读写路径都会先做 owner 过滤与 `// dcp_size` 折叠, 只有重删 CPU 备份 / 恢复路径漏掉了这一步; Issue 还提示即使 id 仍在范围内, 也可能「address the wrong physical row and silently copy/restore incorrect KV data」。PR body 补充: 本 PR 同时是 Draft 验证 PR#38790 的正式落地版本。

实现拆解

1. 抽取统一的 DCP 索引换算入口 (layers/dcp/layout.py): 把原 `HostKVCache.maybe_dcp_kernel_indices` 方法 (位于 `mem_cache/pool_host/base.py`) 改写成模块级纯函数 `maybe_dcp_kernel_indices(indices, dcp_size, dcp_rank)`, 与 `get_dcp_lens`、`filter_dcp_local_kv_indices`、`filter_dcp_local_chunk_kv_indices` 等既有 DCP 索引数学并列, `dcp_size == 1` 时恒等返回。这样做的原因有两个: 一是新调用方 `mem_cache/memory_pool.py` 不属于 HiCache host pool 体系, 方法式 API 无法复用; 二是纯函数把依赖显式化为参数, 避免全局状态耦合。

2. 放宽页对齐断言：原实现带 `assert indices.numel() % self.dcp_size == 0`（要求整加宽页），但重删前缀只有 `seqlen - 1` 个 token、本身是 ragged 的，断言会误杀；新函数只依赖「run 起始页对齐」这一更弱前提，用 `indices[dcp_rank::dcp_size] // dcp_size` 的步长视图选择本 rank 槽位并折叠为物理行。相应地 `pool_host/mla.py` 的 `load_to_device_per_layer` 与 `backup_from_device_all_layer` 改为传入 `self.dcp_size`、`self.dcp_rank`。
3. 修复 PD 重删备份 / 恢复主路径（`mem_cache/memory_pool.py`）：
`MLATokenToKVPool.get_cpu_copy` 与 `load_cpu_copy` 在分块拷贝前各做一次 `maybe_dcp_kernel_indices(indices, self._write_loc_dcp_span, get_parallel().attn_dcp_rank)`。这里用 `_write_loc_dcp_span` 而不是裸 `dcp_size`，是为了沿用池上已有的 `write_loc_is_dcp_resolved` 契约：统一池里 `req_to_token` 存的已是物理行，此时 `span` 为 1、函数透传，天然规避二次换算；备份与恢复各自独立换算，因为恢复阶段 `req_to_token` 里已是重新分配后的新加宽 id，不能复用备份时的物理行。
KDA/Mamba 的 `mamba_indices` 明确不做转换（其索引空间独立）。
4. 测试配套：`test/registered/unit/mem_cache/test_decode_retraction_backup.py` 新增 `TestDcpRetractionBackup`，用 `_bare_mla_pool()` 构造只含 `kv_buffer` 的裸 MLA 池、用 `get_parallel().override(...)` 模拟 `DCP=4/rank=1`，覆盖 `test_restore_lands_on_new_owned_rows`（备份 + 恢复往返落到新的本 rank 行，未涉及行不变）与 `test_resolved_pool_takes_ids_as_rows`（`write_loc_is_dcp_resolved=True` 时 id 原样透传，防双重换算）；`test/registered/unit/mem_cache/test_hicache_dcp_host_pool.py` 改为直接调用纯函数，删除 `_bare_pool` 与随断言一起失效的 `test_ragged_run_is_rejected`。
5. 演进与验证：9 个 commit 几乎全是作者自迭代——先把布尔掩码换成步长视图（1fe7d91），再尝试在两个 `cpu-copy` 调用点内联（6397112、747510c），随后改为共享 helper（dec6560、3d8bd25），最后把测试用例从 `test_retraction_mamba_backup` 迁到 `test_decode_retraction_backup`（60efcfe、40dba53）。CI 侧 `/rerun-test` 覆盖 `test_hicache_dcp_host_pool.py`、`test_decode_retraction_backup.py`、`test_unified_radix_cache_kl_dcp.py`、`test_kimi_linear_pd_dcp4.py`、`test_kimi_linear_pd_dcp4_dspark.py`，五个目标全部通过。

关键文件：

- `python/sglang/srt/mem_cache/memory_pool.py`（模块 内存池；类别 source；类型 core-logic；符号 `MLATokenToKVPool.get_cpu_copy`，`MLATokenToKVPool.load_cpu_copy`）：修复落点：`MLATokenToKVPool.get_cpu_copy` / `load_cpu_copy` 现在在索引物理 `kv_buffer` 前先做 DCP 本地化，并靠 `_write_loc_dcp_span` 兼容统一池的已物理化 id。
- `python/sglang/srt/layers/dcp/layout.py`（模块 索引换算；类别 source；类型 core-logic；符号 `maybe_dcp_kernel_indices`）：新增共享纯函数 `maybe_dcp_kernel_indices`，把「加宽逻辑槽位 -> 本 rank 物理行」的换算从 HiCache host pool 提升为 DCP 索引数学模块的公共能力。
- `test/registered/unit/mem_cache/test_decode_retraction_backup.py`（模块 重删测试；类别 test；类型 test-coverage；符号 `_bare_mla_pool`，`_dcp`，`TestDcpRetractionBackup`，

test_restore_lands_on_new_owned_rows) : 新增 TestDcpRetractionBackup, 用裸 MLA 池 + DCP override 复现「加宽 id 当行号」的两个方向: 往返落到新行、以及已物理化池的透传。

- python/sglang/srt/mem_cache/pool_host/base.py (模块 宿主缓存; 类别 source; 类型 refactor; 符号 HostKVCache.maybe_dcp_kernel_indices) : HostKVCache.maybe_dcp_kernel_indices 被删除 (迁往 layers/dcp/layout.py), 连带移除「整加宽页」断言, 是本 PR 语义变化的来源之一。
- python/sglang/srt/mem_cache/pool_host/mla.py (模块 宿主缓存; 类别 source; 类型 dependency-wiring; 符号 MLATokenToKVPoolHost.load_to_device_per_layer, MLATokenToKVPoolHost.backup_from_device_all_layer) : MLATokenToKVPoolHost 的两个传入口 load_to_device_per_layer、backup_from_device_all_layer 改为调用模块级纯函数并显式传入 self.dcp_size / self.dcp_rank。
- test/registered/unit/mem_cache/test_hicache_dcp_host_pool.py (模块 缓存测试; 类别 test; 类型 test-coverage; 符号 _bare_pool, test_ragged_run_is_rejected, test_matches_owner_rule_on_merged_unordered_pages) : HiCache DCP host pool 单测改为直接调用纯函数, 删除 _bare_pool 与随断言一起失效的 test_ragged_run_is_rejected, 是断言移除的测试侧证据。

关键符号: maybe_dcp_kernel_indices, MLATokenToKVPool.get_cpu_copy, MLATokenToKVPool.load_cpu_copy, MLATokenToKVPoolHost.load_to_device_per_layer, MLATokenToKVPoolHost.backup_from_device_all_layer, TestDcpRetractionBackup.test_restore_lands_on_new_owned_rows, TestDcpRetractionBackup.test_resolved_pool_takes_ids_as_rows

关键源码片段

python/sglang/srt/mem_cache/memory_pool.py

修复落点: MLATokenToKVPool.get_cpu_copy / load_cpu_copy 现在在索引物理 kv_buffer 前先做 DCP 本地化, 并靠 _write_loc_dcp_span 兼容统一池的已物理化 id。

```
# python/sglang/srt/mem_cache/memory_pool.py :: MLATokenToKVPool
```

```
def get_cpu_copy(self, indices, mamba_indices=None, req_pool_index=None):
    # 重删备份: req_to_token 给的是 DCP 加宽逻辑 id, kv_buffer 只存本 rank 的物理行。
    # _write_loc_dcp_span 在统一池已物理化时为 1 (write_loc_is_dcp_resolved),
    # 此时 maybe_dcp_kernel_indices 原样透传, 天然避免二次换算。
    indices = maybe_dcp_kernel_indices(
        indices, self._write_loc_dcp_span, get_parallel().attn_dcp_rank
    )
    current_platform.synchronize()
    kv_cache_cpu = []
    chunk_size = self.cpu_offloading_chunk_size
    for layer_id in range(self.layer_num):
        kv_cache_cpu.append([])
        if self.kv_buffer[layer_id].shape[0] == 0:
            continue
```

```

# 分块拷贝, chunk_indices 这里已经是本 rank 的物理行号
for i in range(0, len(indices), chunk_size):
    chunk_indices = indices[i : i + chunk_size]
    kv_cpu = self.kv_buffer[layer_id][chunk_indices].to(
        "cpu", non_blocking=True
    )
    kv_cache_cpu[-1].append(kv_cpu)
current_platform.synchronize()
return kv_cache_cpu

```

```

def load_cpu_copy(
    self, kv_cache_cpu, indices, mamba_indices=None, req_pool_index=None
):
    # 恢复同样独立换算: 重删后重新分配的是「新的加宽 id」,
    # 不能复用备份时算出的物理行号。KDA/Mamba 状态的 mamba_indices 不做转换。
    indices = maybe_dcp_kernel_indices(
        indices, self._write_loc_dcp_span, get_parallel().attn_dcp_rank
    )
    current_platform.synchronize()
    chunk_size = self.cpu_offloading_chunk_size
    for layer_id in range(self.layer_num):
        if self.kv_buffer[layer_id].shape[0] == 0:
            continue
        for i in range(0, len(indices), chunk_size):
            chunk_indices = indices[i : i + chunk_size]
            kv_cpu = kv_cache_cpu[layer_id][i // chunk_size]
            # 换算对称性的显式检查: 备份与恢复两侧必须得到同样多的行
            assert kv_cpu.shape[0] == len(chunk_indices)
            kv_chunk = kv_cpu.to(self.kv_buffer[0].device, non_blocking=True)
            self.kv_buffer[layer_id][chunk_indices] = kv_chunk
        current_platform.synchronize()

```

python/sglang/srt/layers/dcp/layout.py

新增共享纯函数 maybe_dcp_kernel_indices, 把「加宽逻辑槽位 -> 本 rank 物理行」的换算从 HiCache host pool 提升为 DCP 索引数学模块的公共能力。

```

# python/sglang/srt/layers/dcp/layout.py
# 该模块原本只放 DCP 的纯索引数学 (get_dcp_lens、filter_dcp_local_kv_indices 等)。
# 本 PR 把原本是 HostKVCache 方法的换算提升到这里,
# 让 HiCache host pool 与 MLA 池的重删 CPU 备份 / 恢复共用同一份语义。

```

```

def maybe_dcp_kernel_indices(
    indices: torch.Tensor, dcp_size: int, dcp_rank: int
) -> torch.Tensor:
    """Widened logical slots -> this rank's physical rows.

```

Owner rule: $\text{slot} \% \text{dcp_size} == \text{dcp_rank}$, $\text{row} = \text{slot} // \text{dcp_size}$. The run starts page-aligned, so a strided view selects the owned slots without a mask.

```

"""
if dcp_size == 1:
    return indices # DCP=1 是恒等情形，直接返回原张量避免多余拷贝
# 关键取舍：去掉原实现的「必须整加宽页」断言（重删前缀是 seqlen - 1，天然 ragged），
# 只依赖更弱的前提「run 起始页对齐」，用步长视图代替布尔掩码：
# 选中 indices[dcp_rank]、indices[dcp_rank + dcp_size]、...，再整体 // dcp_size
return indices[dcp_rank::dcp_size] // dcp_size

```

```

# 同模块中的对照实现：读取侧只做选择、仍保持加宽 id，由调用方再折叠
# def filter_dcp_local_kv_indices(kv_indices):
# parallel = get_parallel()
# if parallel.dcp_enabled:
# kv_indices = kv_indices[kv_indices % parallel.dcp_size == parallel.dcp_rank]
# return kv_indices

```

test/registered/unit/mem_cache/test_decode_retraction_backup.py

新增 TestDcpRetractionBackup，用裸 MLA 池 + DCP override 复现「加宽 id 当行号」的两个方向：往返落到新行、以及已物理化池的透传。

```

# test/registered/unit/mem_cache/test_decode_retraction_backup.py
DCP_SIZE = 4
DCP_RANK = 1
DCP_ROWS = 8

```

```

def _bare_mla_pool() -> MLATokenToKVPool:
    # 绕过构造函数：只挂上 kv_buffer / layer_num / 分块大小，
    # 每行以 arange + 100 * layer 填充，方便断言「KV 被搬到了哪一行」。
    pool = object.__new__(MLATokenToKVPool)
    pool.layer_num = 2
    pool.cpu_offloading_chunk_size = 3
    pool.kv_buffer = [
        (torch.arange(DCP_ROWS, dtype=torch.float32) + 100 * layer).view(DCP_ROWS, 1, 1)
        for layer in range(pool.layer_num)
    ]
    return pool

```

```

def _dcp():
    # 通过 runtime_context 覆盖并行配置，模拟 DCP=4 且本 rank 为 1
    return get_parallel().override(
        dcp_enabled=True, attn_dcp_size=DCP_SIZE, attn_dcp_rank=DCP_RANK
    )

```

```

class TestDcpRetractionBackup(unittest.TestCase):
    """`req_to_token` names KV slots in the widened DCP id space while
    `kv_buffer` holds only this rank's rows; a widened id used as a row index
    """

```

```
reads past the buffer or copies another token's row."""
```

```
def test_restore_lands_on_new_owned_rows(self):
    pool = _bare_mla_pool()
    before = [buf.clone() for buf in pool.kv_buffer]
    old_widened = torch.arange(0, 12, dtype=torch.int64) # 备份来源的加宽 id
    new_widened = torch.arange(12, 24, dtype=torch.int64) # 重删后新分配的加宽 id

    with _dcp():
        # 备份与恢复各自独立换算：旧 id 读出本 rank 行，写回新 id 对应的本 rank 行
        pool.load_cpu_copy(pool.get_cpu_copy(old_widened), new_widened)

    old_rows = old_widened[DCP_RANK::DCP_SIZE] // DCP_SIZE
    new_rows = new_widened[DCP_RANK::DCP_SIZE] // DCP_SIZE
    self.assertEqual(new_rows.tolist(), [3, 4, 5])
    untouched = torch.tensor(
        [r for r in range(DCP_ROWS) if r not in new_rows.tolist()]
    )
    for layer in range(pool.layer_num):
        torch.testing.assert_close(
            pool.kv_buffer[layer][new_rows], before[layer][old_rows]
        ) # 数据确实搬到了新的本 rank 行
        torch.testing.assert_close(
            pool.kv_buffer[layer][untouched], before[layer][untouched]
        ) # 未涉及的行保持原样，排除误写

def test_resolved_pool_takes_ids_as_rows(self):
    pool = _bare_mla_pool()
    pool.write_loc_is_dcp_resolved = True # 统一池：req_to_token 中已是物理行
    rows = torch.arange(DCP_ROWS, dtype=torch.int64)

    with _dcp():
        kv_cpu = pool.get_cpu_copy(rows)

    for layer in range(pool.layer_num):
        # 双重换算防护：DCP 打开时也要原样透传
        torch.testing.assert_close(torch.cat(kv_cpu[layer]), pool.kv_buffer[layer])
```

评论区精华

本 PR 的评审讨论非常轻量：[review_comments_count = 0](#)，唯一一条 review 来自 ShangmingCai 的 APPROVED —— 「This minor fix looks good to me.」。Issue 线程中的两条评论都是流程性的：作者用 [/rerun-test](#) 指定 5 个测试文件，github-actions[bot] 回报 ubuntu-latest、1-gpu-5090、4-gpu-b200、8-gpu-b200 四个 runner 上的用例全部 [🔗](#)。真正有价值的「讨论」发生在 PR body 与 commit message 里：作者主动说明了断言被删的原因（重删前缀是 [seqlen - 1](#) 的 ragged run）、[_write_loc_dcp_span](#) 与 [write_loc_is_dcp_resolved](#) 的契约、以及一个刻意不修的问题——强制重删下约 4/48 的恢复请求会退化 token 重复，且 [--dcp-size 1](#) 同样复现，因此判定为独立问题而非 DCP 缺陷。

- 索引换算逻辑该放在哪一层 (design): 确定为 layers/dcp/layout.py 的模块级纯函数 maybe_dcp_kernel_indices(indices, dcp_size, dcp_rank), dcp_size/rank 显式传参, HiCache host pool 与 MLA 池共用。
- 删除「整加宽页」断言的正确性边界 (correctness): 断言被接受移除, 换取重删路径可用; 代价是函数对「起始对齐」的假设从强制校验退化为文档契约, 未来误用会静默选错槽位。
- 统一池上的双重换算防护 (design): 以池属性作为开关, 单一换算点同时服务「加宽 id」与「已物理化 id」两种池, 避免二次折叠。
- DCP 回归用例的归属测试文件 (testing): 最终落在 test/registered/unit/mem_cache/test_decode_retraction_backup.py 的 TestDcpRetractionBackup, 与既有的 MHA/draft 备份用例共用同一裸池构造风格。
- 评审结论与验证强度 (other): 代码评审轻量通过; 深度验证证据主要来自 PR body 中的 4xB200 实测 (main 约 12 次重删即 assert, 本分支 140-200 次重删无 assert)。

风险与影响

- 风险:
 1. 静默错误行风险 (最高): 把加宽 id 当行号使用时, 只要 id 落在此 rank 的行范围内就不会崩溃, 而是复制 / 写回错误 token 的 KV (Issue#38645 原文明确警告)。本 PR 修掉了该路径, 但此类错误缺乏端到端可观测性, 主要靠 test_restore_lands_on_new_owned_rows 这类单测与 Kimi-Linear-48B 的 logprob 对比守护。
 2. 页对齐断言移除的边界: 新函数删掉了 indices.numel() % dcp_size == 0 的 fail-fast 断言, 改为口头契约「run starts page-aligned」。若未来有调用方传入起始未对齐的 run (例如带 offset 的 chunk 视图), 步长视图会静默选错槽位而不报错; test_hicache_dcp_host_pool.py 中原本守护这一点的 test_ragged_run_is_rejected 也被删除, 回归网出现缺口。
 3. 对池契约的隐式依赖: get_cpu_copy / load_cpu_copy 的正确性依赖 _write_loc_dcp_span (配合 write_loc_is_dcp_resolved) 与实际索引空间一致。MLATokenToKVPool 的子类 (如 MLATokenToKVPoolFP4) 若未正确维护该属性, 会出现误换算或属性缺失; get_parallel().attn_dcp_rank 取的是全局并行上下文而非池自身 rank, 两者不一致时会算错行。
 4. 未覆盖范围: Mamba/KDA 的 mamba_indices 不做转换, load_cpu_copy 中「恢复后重复 token 退化」问题被明确留作后续; assert kv_cpu.shape[0] == len(chunk_indices) 是换算对称性的唯一显式检查, 一旦备份 / 恢复两侧 span 判断不同会在此处断言失败而非静默错位。
 5. CI 信号混杂: PR body 中的 CI 状态槽位显示 PR Test (Base)/(Extra)/(AMD ROCm 10) 三个 job 为 🟡, 而针对性 rerun 的 5 个测试全绿, 合并前需要确认失败项与本次变更无关。- 影响: 用户侧: PD 分离 + DCP > 1 + MLA (Kimi 系列) 的 decode 节点在容量压力触发重删时不再崩溃, 请求可继续完成; 作者在 4xB200 上实测 main 分支约 12 次重删即 assert, 本分支 140-200 次重删无 assert, per-token logprob 与不重删参考值在噪声范围内。系统侧: layers/dcp/layout.py 成为 DCP 索引换算的公共入口, HiCache host pool 与 KV 池的语义一致性问题被收敛到单点; memory_pool.py 处于热路径上游, 但只在备份 / 恢复时执行, 对正常 decode 吞吐无影响 (转发路径不受影响)。

团队侧：这是「加宽逻辑索引 vs rank 物理行」这一 DCP 核心约定的一次契约补齐，后续任何新增访问 kv_buffer 的旁路路径都应先过 maybe_dcp_kernel_indices。测试层面新增的 TestDcpRetractionBackup 提供了可复用的裸池构造方法（_bare_mla_pool + _dcp），便于继续补 DCP 场景用例。 - 风险标记：静默错行风险，页对齐断言移除，双重换算依赖池契约，KDA/Mamba 状态未覆盖，已知 token 重复退化未修，PR 主 CI 槽位显示失败

关联脉络

- PR #38790 [BugFix][DCP] Localize MLA KV indices for retraction CPU backup/restore: 同一 Issue#38645 的 Draft 验证 PR，本 PR body 明确说明是其正式落地版本；两者修复目标、索引语义与验证矩阵（v0.5.17 A/B、v0.5.19 容量边界扫描）完全对应。
- PR #35233 [AMD] Fix registered HiCache host pointer aliases: 同样触及 HiCache host pool 的指针 / 索引域问题（pool_host/common.py、pool_host/mla.py 等），与本 PR 同属「host pool 与 device/ 逻辑索引空间错配」问题族。
- PR #38941 [Fix] Merge adjacent KV-row frees so a mid-page split under DCP cannot double-free: 同为 DCP 场景下 KV 行索引 / 页内接缝的正确性修复，改动 mem_cache 公共层（base_prefix_cache.py、common.py），与本 PR 一起构成 DCP KV 索引语义的补强序列。