PR #35762 完整报告

sgl-project/sglang

[PD] Pack DCP1→DCP-N PD KV transfers into dest-contiguous RDMA blocks

合并时间：2026-08-29 12:41

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/35762

执行摘要

- 一句话：DCP1→N 传输打包成连续 RDMA 块，TTFT 最高降 39%
- 推荐动作：值得精读。核心看点：把 descriptor-bound 的传输瓶颈重新建模为带宽问题；try_pack_dcp_src 的 stream 同步与 fallback 设计；NIXL 异步读取与 chunk barrier 的配合方式；以及按 DCP rank 固定分区来支持并发异步提交的思路。建议重点阅读 dcp_pack.py 与 nixl/conn.py::transfer_worker 两处，理解 buffer 生命周期与同步语义后再评估是否移植到 DSPARK。

功能与动机

PR body 明确指出：DCP 按 token 轮询分配 KV 所有权（owned_offsets = arange(rank, num_kv_tokens, dcp_size)），因此 DCP1→DCP-N 传输的源索引是 strided 的，group_concurrent_contiguous 无法合并这些索引：relayout 路径每个 token 每层发出一个 RDMA 描述符（tokens x layers per chunk），传输变成描述符受限而非带宽受限。目标是把 owned MLA 行先在 prefill 侧 gather 成连续块，使 Mooncake/NIXL 能发送 page 级或更大的块。

实现拆解

整个变更分为五步：

1. 新增公共打包模块：新建 python/sglang/srt/disaggregation/common/dcp_pack.py，提供三个核心函数：dcp_pack_buffer_bytes() 按 dcp_size x ceil(max_tokens / dcp_size) x sum(token_item_lens) 计算 buffer 尺寸；try_pack_dcp_src() 在专用 gather stream 上调用 Triton 内核把 strided 源行拷入 pack buffer，并在空间不足时返回 None 以回退到逐 token RDMA；init_dcp_pack_buffers() 按 transfer worker 数量分配并注册 StagingBuffer，每个 DCP rank 拥有固定区域。
2. 新增 Triton gather 内核：新建 python/sglang/kernels/ops/kvcache/pd_dcp_gather.py，提供 copy_mla_rows_into_pack()，直接消费 kv_data_ptrs（裸指针），并在 python/sglang/kernels/ops/kvcache/__init__.py 导出，保持 KVArgs 后端中立。
3. StagingBuffer 增加 gather stream：修改 python/sglang/srt/disaggregation/common/staging_buffer.py，懒创建并复用 get_gather_stream()，打包前 wait_stream(torch.cuda.default_stream(...))，打包后 synchronize()，复用既有 staging 同步路径。

4. 统一公共入口：在 python/sglang/srt/disaggregation/common/conn.py 的 CommonKVManager 增加 _register_staging_memory() 抽象方法（默认抛 NotImplementedError）和 _init_dcp_pack_buffers_once()（peer 注册时初始化一次）。NIXL 与 Mooncake 各自实现 _register_staging_memory() 并接入 transfer worker。
5. 后端集成与测试配套：NIXL 侧 transfer_worker() 为每个 DCP rank 调用 _pack_dcp_rank_once() 只打包一次，异步提交后依赖既有 chunk barrier 再 repack；Mooncake 侧 send_kvcache_dcp() 增加 pack_buffer 参数，打包成功则直接发送 packed blocks。测试方面新增 test/registered/unit/disaggregation/test_dcp_pack.py（描述符合并、buffer 尺寸、打包偏移与稠密索引）、test/registered/kernels/ops/kvcache/test_pd_dcp_gather.py（gather 内核）、并在 test/registered/unit/disaggregation/test_nixl_backend_basic.py 增加"多个 DCP 目的端在 chunk barrier 前使用互斥 pack 区域"的集成测试；精度与端到端由 test/registered/disaggregation/test_kimi_linear_pd_dcp4.py 覆盖。

关键文件：

- python/sglang/srt/disaggregation/common/dcp_pack.py（模块 打包层；类别 source；类型 core-logic；符号 dcp_pack_buffer_bytes, try_pack_dcp_src, init_dcp_pack_buffers）：新增核心打包模块，实现 pack buffer 尺寸计算、Triton gather 打包与 buffer 初始化，是整条打包路径的基石。
- python/sglang/srt/disaggregation/nixl/conn.py（模块 传输后端；类别 source；类型 core-logic；符号 _register_staging_memory, transfer_worker, _pack_dcp_rank_once）：NIXL 传输后端的核心集成点：transfer_worker 增加 worker_index 并按 DCP rank 打包一次，异步提交后依赖 chunk barrier 复用 buffer。
- python/sglang/srt/disaggregation/mooncake/conn.py（模块 传输后端；类别 source；类型 core-logic；符号 _register_staging_memory, send_kvcache_dcp, transfer_worker）：Mooncake 传输后端的集成：send_kvcache_dcp 增加 pack_buffer 参数，打包成功后直接发送连续块，transfer_worker 传入 worker 私有 buffer。
- python/sglang/srt/disaggregation/common/conn.py（模块 公共层；类别 source；类型 dependency-wiring；符号 _register_staging_memory, _init_dcp_pack_buffers_once）：公共管理类增加 _dcp_pack_buffers 状态、_register_staging_memory 抽象与 _init_dcp_pack_buffers_once，统一两个后端的初始化入口。
- python/sglang/srt/disaggregation/common/staging_buffer.py（模块 暂存层；类别 source；类型 core-logic；符号 get_gather_stream）：StagingBuffer 新增懒创建并复用的 gather stream，让 DCP 打包与既有 staging 同步路径保持一致。
- python/sglang/kernels/ops/kvcache/pd_dcp_gather.py（模块 内核层；类别 infra；类型 infrastructure；符号 _copy_mla_rows_into_pack_kernel, copy_mla_rows_into_pack）：新增 Triton gather 内核，直接从裸指针 gather strided MLA 行到 pack buffer，是打包路径的计算核心。
- test/registered/unit/disaggregation/test_dcp_pack.py（模块 打包测试；类别 test；类型 test-coverage；符号 TestPackedDcpGrouping, test_packed_groups_collapse_cyclic_src, TestDcpPackBufferBytes, test_sizes_fixed_regions_for_each_dcp_rank）：新增单元测试，覆盖打包后描述符合并（collapse cyclic src）、buffer 尺寸、非法 kv_item_lens 拒绝、打包偏移与稠密索引。

- test/registered/unit/disaggregation/test_nixl_backend_basic.py（模块 传输测试；类别 test；类型 test-coverage；符号 test_dcp_destinations_use_disjoint_pack_regions_before_chunk_barrier, send_kvcache_dcp, check_xfer_state）：新增 NIXL 集成测试，验证多个 DCP 目的端在 chunk barrier 前使用互斥 pack 区域，且同一 DCP rank 复用同一 packed source。

关键符号：try_pack_dcp_src, init_dcp_pack_buffers, dcp_pack_buffer_bytes, copy_mla_rows_into_pack, _copy_mla_rows_into_pack_kernel, get_gather_stream, _init_dcp_pack_buffers_once, _register_staging_memory, transfer_worker, send_kvcache_dcp, _pack_dcp_rank_once

关键源码片段

python/sglang/srt/disaggregation/common/dcp_pack.py

新增核心打包模块，实现 pack buffer 尺寸计算、Triton gather 打包与 buffer 初始化，是整条打包路径的基石。

```
# python/sglang/srt/disaggregation/common/dcp_pack.py
# 把 cyclic DCP ownership 造成的 strided 源行先打包成连续块，
# 使 group_concurrent_contiguous 能坍缩成少量大块 RDMA 描述符。

def try_pack_dcp_src(
    *,
    pack_buffer: StagingBuffer,
    kv_data_ptrs: Sequence[int],
    src_token_indices: npt.NDArray[np.integer],
    token_item_lens: Sequence[int],
    pack_offset_bytes: int = 0,
) -> Optional[Tuple[List[int], npt.NDArray[np.int64]]]:
    # 预算不足时回退到逐 token RDMA，保证功能不因显存紧张而中断
    n = int(src_token_indices.size)
    if n == 0:
        empty = np.empty((0,), dtype=np.int64)
        return [], empty
    required = n * sum(int(item_len) for item_len in token_item_lens)
    required_end = pack_offset_bytes + required
    if not pack_buffer.fits(required_end):
        logger.warning(
            "PD DCP pack buffer too small for byte range [%s, %s) (have %s); "
            "falling back to per-token RDMA",
            pack_offset_bytes,
            required_end,
            pack_buffer.get_size(),
        )
        return None

    # 在专用 gather stream 上等待默认流写完 KV，再执行 Triton gather，
    # 完成后同步，确保传输读取前数据就绪
```

```
    pack = pack_buffer.buffer.narrow(0, pack_offset_bytes, required)
    row_indices = torch.as_tensor(
        src_token_indices, device=pack.device, dtype=torch.int64
    )
    gather_stream = pack_buffer.get_gather_stream()
    gather_stream.wait_stream(torch.cuda.default_stream(pack.device))
    with torch.cuda.stream(gather_stream):
        copy_mla_rows_into_pack(kv_data_ptrs, row_indices, pack, token_item_lens)
    gather_stream.synchronize()

    # 返回连续块内各 layer 的起始指针与稠密索引 [0, n),
    # 传输侧只需把指针交给后端按连续区间发送
    packed_ptrs: List[int] = []
    offset = 0
    base = pack_buffer.get_ptr() + pack_offset_bytes
    for item_len in token_item_lens:
        packed_ptrs.append(base + offset)
        offset += n * int(item_len)
    return packed_ptrs, np.arange(n, dtype=np.int64)


def init_dcp_pack_buffers(
    register_fn,
    kv_args,
    count: int,
    dcp_size: int,
) -> List[StagingBuffer]:
    # 大小 = dcp_size x ceil(max_tokens / dcp_size) x sum( 每层 token 字节 ),
    # 每个 DCP rank 拥有固定区域，保证 NIXL 异步读取期间不会互相覆盖
    max_tokens = max_prefill_buffer_tokens()
    if max_tokens <= 0:
        max_tokens = get_schedule().max_prefill_tokens
    size_bytes = dcp_pack_buffer_bytes(
        kv_args.kv_item_lens, kv_args.page_size, max_tokens, dcp_size
    )
    device = f"cuda:{kv_args.gpu_id}"
    custom_mem_pool, _ = _get_custom_mem_pool(device)

    buffers = []
    for _ in range(count):
        buf = StagingBuffer(size_bytes, device, kv_args.gpu_id, custom_mem_pool=custom_mem_
        pool)
        register_fn(buf.get_ptr(), buf.get_size())
        buffers.append(buf)
    logger.info(
        "PD DCP pack buffers allocated: %d x %.1f MB (max_tokens=%d)",
        count,
        size_bytes / (1024 * 1024),
        max_tokens,
```

```
    )
    return buffers
```

test/registered/unit/disaggregation/test_dcp_pack.py

新增单元测试，覆盖打包后描述符合并（collapse cyclic src）、buffer 尺寸、非法 kv_item_lens 拒绝、打包偏移与稠密索引。

```
class TestPackedDcpGrouping(CustomTestCase):
    def test_packed_groups_collapse_cyclic_src(self):
        # 复现 DCP cyclic ownership 的典型场景：dcp_size = 4 时，
        # rank 0 拥有第 0、4、8 ... 个 token，源页分布是 strided 的
        page_size = 64
        dcp_size = 4
        src_pages = np.arange(4, dtype=np.int32)
        dst_pages = np.array([7], dtype=np.int32)
        plan = build_dcp_token_transfer_plan(
            src_pages,
            dst_pages,
            physical_page_size=page_size,
            dcp_size=dcp_size,
            dcp_rank=0,
            num_kv_tokens=256,
        )
        # 未打包时：64 个源索引，每个都落在独立并发组里，
        # 即每 token 每层 1 个 RDMA 描述符，正是本 PR 要消除的形态
        raw_src, _ = group_concurrent_contiguous(
            plan.src_token_indices, plan.dst_token_indices
        )
        self.assertEqual(len(raw_src), 64)
        self.assertTrue(all(len(group) == 1 for group in raw_src))

        # 打包后：源索引变成稠密 [0, 64)，目的索引仍是同一段连续区间，
        # 因而坍缩成单个组，一次 RDMA 即可覆盖全部 token
        packed_src = np.arange(plan.dst_token_indices.size, dtype=np.int64)
        packed_groups, _ = group_concurrent_contiguous(
            packed_src, plan.dst_token_indices
        )
        self.assertEqual(len(packed_groups), 1)
        self.assertEqual(len(packed_groups[0]), 64)
```

评论区精华

review 评论不多但有两处值得注意：

- ShangmingCai（APPROVED）："Looks good to me, very clean. cc: @YAMY1234 for staging_buffer change review." 说明 staging_buffer.py 作为公共组件被单独拉出来交叉审查。
- YAMY1234（APPROVED）："Staging buffer change LGTM!" 确认 gather stream 复用与既有 staging 同步路径一致。

- 其余 issue 评论集中在 CI：kpham-sgl 多次 /rerun-test test_kimi_linear_pd_dcp4.py，ShangmingCai 执行 /rerun-group disaggregation 与 /tag-and-rerun-ci，github-actions 全部 😕，最终 kpham-sgl 确认 "Passed all based CI and PD disaggs test" 后合并。
- staging_buffer 公共组件变更评审 (design): YAMY1234 确认 Staging buffer change LGTM，无阻塞项。
- PD DCP 端到端测试重跑与验收 (testing): kpham-sgl 确认 Passed all based CI and PD disaggs test，随后合并。

风险与影响

- 风险：
  1. 核心路径变更且默认启用：打包路径没有 rollout toggle（commit 明确 "Enable packed transfers by default"），所有 DCP1→DCP-N 的 MLA 传输都走新路径；虽然 try_pack_dcp_src() 在 buffer 不足时会 fallback 到逐 token RDMA，但该 fallback 不覆盖 Triton 内核本身的正确性问题，回归影响面较大。
  2. 显存成本增加：按代码注释，32768 tokens、61 层 MLA 576 bf16 时每个 pack buffer 约 2.14 GiB，4 个 worker 即 8.58 GiB/rank；init_dcp_pack_buffers() 按 max_prefill_buffer_tokens() 与 get_schedule().max_prefill_tokens 取上限，长 prefill/ 大 chunk 配置下显存占用可能进一步放大。
  3. NIXL 异步语义耦合：NIXL 异步提交 pack 区域，依赖既有 chunk barrier 保证 repack 前传输完成；若 barrier 行为变化，可能引入 source-buffer reuse race（新测试 test_dcp_destinations_use_disjoint_pack_regions_before_chunk_barrier 已覆盖，但仍是时序敏感逻辑）。
  4. 接口签名变更：_register_staging_memory() 去掉 gpu_id 参数改为读取 self.kv_args.gpu_id，所有调用点与测试需同步（最后一个 commit 即为修复该签名不匹配）。
  5. 覆盖缺口：DSPARK 未走打包路径（PR body 明示留作 follow-up）；小请求（ISL 4096/conc 8）几乎无收益，说明打包开销在该区间未能体现价值。 - 影响：用户侧：PD 分离 + Prefill DCP1→Decode DCP-N + MLA 的部署（如 Kimi-Linear、GLM 类）在 NIXL 后端 TTFT 降 28%~39%、输出吞吐提升最高 60.7%；Mooncake 后端也有 9%~14% 的 TTFT 改善，且两个后端在 ISL≥32k 时收敛到约 3% 内。对 DCP1→DCP1 或非 MLA 场景无影响。系统侧：每个 prefill TP rank 增加若干 pack buffer 显存开销；transfer worker 线程新增 worker_index 参数以索引私有 buffer；公共 StagingBuffer 的 gather stream 语义成为传输同步的一部分。团队侧：公共层 CommonKVManager / StagingBuffer 的改动已做交叉审查（YAMY1234），后续 DSPARK 打包与 _register_staging_memory 签名收敛需要跟踪。
- 风险标记：核心路径变更，默认启用打包路径，显存成本增加，NIXL 异步依赖 chunk barrier，DSPARK 未覆盖，小 chunk 收益有限

关联脉络

- PR #36934 [Fix] Drop the duplicated DSpark draft sample_block call: 本 PR body 明确 DSPARK 打包留作 follow-up，而 DSpark 路径近期也有修复，未来可能复用 dcp_pack 思路。
- PR #36852 [ROCm][Bugfix] Use token-level KV indices in the aiter ASM context-prefill gather: 同属 KV gather/ 注意力后端路径，关注 gather 索引正确性，说明 gather 形态是近期持续修缮的主线。
- PR #36704 Refactor JIT kernel and expert-pack directory layout: 本 PR 新增 Triton kernel 落位于重构后的 python/sglang/kernels/ops/kvcache/ 目录，与 JIT kernel 布局演进方向一致。