PR #31574 完整报告

sgl-project/sglang

[EPD] Batch embedding cache host-device range copies

合并时间：2026-08-14 22:01

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/31574

执行摘要

- 一句话：embedding 缓存 H2D/D2H 批量直拷，碎片吞吐提升约 3-5 成
- 推荐动作：值得精读。transfer.cu 中 cudaMemcpyBatchAsync 的动态探测 + ABI 分派 + 多级回退模式，可作为 sgl-kernel 新增 CUDA 算子的参考实现；embedding_cache_controller.py 的「计划构建与执行分离」也值得在缓存迁移场景复用。关注 embedding cache/EPD 的工程师建议按 key_files 顺序阅读，重点看 _copy_embedding_page_runs 与 transfer_embedding_ranges_direct 的完整实现。

功能与动机

PR body 明确指出痛点：The paged multimodal embedding cache may store one embedding across multiple non-contiguous pinned-host page runs. Loading from or storing to this pool currently submits one Tensor.copy_ operation per run from Python. As the cache becomes fragmented, repeated Python dispatch and CUDA copy submission reduce effective H2D and D2H throughput. 同时说明为什么不能复用既有内核：Existing KV transfer kernels are tied to KV layouts and index-based gather/scatter semantics, so they are not a good fit for copying contiguous embedding ranges. 因此需要一个面向 embedding 连续范围的批量拷贝算子。

实现拆解

1. 新增 sgl-kernel 批量直拷算子：在 python/sglang/kernels/aot/csrc/kvcacheio/transfer.cu 新增 transfer_embedding_ranges_direct，先把 src_starts/dst_starts/lengths 换算成按行字节对齐的指针与字节数，完整校验所有范围后再调用 cudaMemcpyBatchAsync 一次性提交；针对 CUDA 13.0 的入参签名变化（新增 fail_idx 出参）用 dlsym 动态探测符号并按 runtime 版本分派，同时在编译期排除 ROCm/MUSA/ 旧 CUDA，运行时对 NULL 流、驱动版本不足、cudaErrorNotSupported/cudaErrorCallRequiresNewerDriver 等场景回退到逐条 cudaMemcpyAsync。配套在 sgl_kernel_ops.h 声明、common_extension.cc 注册 Torch schema，并在 python/sgl_kernel/kvcacheio.py 暴露 Python 封装。
2. 拆分缓存控制器中的两类拷贝计划：embedding_cache_controller.py 将原 build_transfer_buffers 更名为 _build_storage_transfer_buffers（仍服务于 Mooncake 存储 GET/PUT 的「指针 + 字节大小」），新增 _build_host_device_transfer_plan 生成 token 级 src_starts/dst_starts/lengths，以 src_is_pool 区分 H2D/D2H 方向并支持 dst_token_offset。

3. 统一 H2D/D2H 拷贝入口：新增 copy_embedding_page_runs，在存在 CUDA 一侧且算子可用时调用批量直拷，否则回退到原有 copy 循环；load_to_device_async 与 _copy_tensor_to_pool 均改为调用该入口，AsyncCopyHandle 的事件 / 等待语义保持不变。模块导入处用 try/except ImportError 与 torch.ops.sgl_kernel 属性探测双重判断算子可用性。
4. 测试配套：test/registered/unit/mem_cache/test_embedding_cache_controller.py 覆盖 build_storage_transfer_buffers 重命名后行为与碎片化 entry 的 H2D/D2H 计划期待值；python/sglang/kernels/aot/tests/test_kvcacheio.py 新增 test_transfer_embedding_ranges_direct，在非默认 stream 上验证 H2D/D2H 批量拷贝与 copy 参考一致（含最后一个范围不完整的碎片形态）。PR body 另附完整 benchmark 脚本与手动 EPD 双请求缓存命中验证（第二轮 Local Hits=1）。

关键文件：

- python/sglang/srt/mem_cache/embedding_cache_controller.py（模块 缓存控制器；类别 source；类型 entrypoint；符号 build_transfer_buffers, _build_storage_transfer_buffers, _build_host_device_transfer_plan, _copy_embedding_page_runs）：变更入口：将 build_transfer_buffers 拆分为存储缓冲与 host-device 拷贝计划两类构建逻辑，新增统一拷贝入口 _copy_embedding_page_runs 并接入 load_to_device_async 与 _copy_tensor_to_pool，带算子可用性探测与 Python 回退。
- python/sglang/kernels/aot/csrc/kvcacheio/transfer.cu（模块 传输内核；类别 other；类型 core-logic；符号 transfer_embedding_ranges_direct）：CUDA 批量拷贝算子实现核心：cudaMemcpyBatchAsync 一次提交、CUDA 12.8/13.0 ABI 分派、驱动 / 符号探测与多级回退，以及提交前完整计划校验。
- python/sglang/kernels/aot/python/sgl_kernel/kvcacheio.py（模块 算子封装；类别 source；类型 core-logic；符号 transfer_embedding_ranges_direct）：向 Python 暴露新算子的薄封装，供缓存控制器调用；保持与既有 transfer_kv 系列一致的封装风格。
- test/registered/unit/mem_cache/test_embedding_cache_controller.py（模块 控制器测试；类别 test；类型 test-coverage；符号 test_build_h2d_copy_plan_for_fragmented_entry, test_build_d2h_copy_plan_for_fragmented_entry）：覆盖 _build_storage_transfer_buffers 重命名后的行为，并新增碎片化 entry 的 H2D/D2H 拷贝计划期待值测试。
- python/sglang/kernels/aot/tests/test_kvcacheio.py（模块 内核测试；类别 test；类型 test-coverage；符号 ref_copy_embedding_ranges, test_transfer_embedding_ranges_direct）：新增 CUDA 正确性测试：非默认流上验证 H2D/D2H 批量拷贝与 Tensor.copy_ 参考一致，覆盖碎片范围与不完整尾段。
- python/sglang/kernels/aot/include/sgl_kernel_ops.h（模块 内核声明；类别 source；类型 core-logic；符号 transfer_embedding_ranges_direct）：新增 transfer_embedding_ranges_direct 的 C++ 声明，保持 aot 头文件与实现同步。
- python/sglang/kernels/aot/csrc/common_extension.cc（模块 算子注册；类别 source；类型 core-logic；符号 transfer_embedding_ranges_direct）：注册 transfer_embedding_ranges_direct 的 Torch library schema 与 CUDA dispatch，使

Python 侧可通过 torch.ops.sgl_kernel 调用。

关键符号：transfer_embedding_ranges_direct, _copy_embedding_page_runs, _build_host_device_transfer_plan, _build_storage_transfer_buffers, load_to_device_async, _copy_tensor_to_pool, ref_copy_embedding_ranges, test_transfer_embedding_ranges_direct

关键源码片段

python/sglang/srt/mem_cache/embedding_cache_controller.py

变更入口：将 build_transfer_buffers 拆分为存储缓冲与 host-device 拷贝计划两类构建逻辑，新增统一拷贝入口 _copy_embedding_page_runs 并接入 load_to_device_async 与 _copy_tensor_to_pool，带算子可用性探测与 Python 回退。

```
# 模块级导入：优先使用 sgl_kernel 的批量直拷算子，缺失或未注册时降级为 None
try:
    from sgl_kernel.kvcacheio import transfer_embedding_ranges_direct
except ImportError:
    transfer_embedding_ranges_direct = None

if transfer_embedding_ranges_direct is not None and not hasattr(
    torch.ops.sgl_kernel, "transfer_embedding_ranges_direct"
):
    transfer_embedding_ranges_direct = None


def _build_host_device_transfer_plan(
    entry: EmbeddingCacheEntry,
    pool: EmbeddingPool,
    src_is_pool: bool,
    dst_token_offset: int = 0,
) -> Tuple[List[int], List[int], List[int]]:
    """把一条 embedding 的多个 page run 展开成 token 级拷贝计划。

    src_is_pool 决定 pool 侧是源还是目标，dst_token_offset 支持写入设备侧
    的某个偏移位置。返回 (src_starts, dst_starts, lengths) 三个等长列表，
    交给批量拷贝算子一次提交。
    """
    src_starts: List[int] = []
    dst_starts: List[int] = []
    lengths: List[int] = []
    copied = 0

    for run in entry.page_runs:
        valid_tokens = min(pool.page_size * run.length, entry.num_tokens - copied)
        if valid_tokens <= 0:
            break

        pool_start = run.start * pool.page_size
```

```
            if src_is_pool:
                src_starts.append(pool_start)
                dst_starts.append(dst_token_offset + copied)
            else:
                src_starts.append(copied)
                dst_starts.append(pool_start)
            lengths.append(valid_tokens)
            copied += valid_tokens

        return src_starts, dst_starts, lengths


    def _copy_embedding_page_runs(
        self,
        src: torch.Tensor,
        dst: torch.Tensor,
        entry: EmbeddingCacheEntry,
        pool: EmbeddingPool,
        src_is_pool: bool,
        dst_token_offset: int = 0,
    ) -> None:
        """H2D/D2H 的统一拷贝入口：优先批量直拷，不支持时回退到 copy_ 循环。"""
        src_starts, dst_starts, lengths = _build_host_device_transfer_plan(
            entry, pool, src_is_pool, dst_token_offset
        )
        if not lengths:
            return

        # 只有涉及 CUDA 且算子可用时才走 C++ 路径，纯 CPU/ 存储路径保持原有行为
        has_cuda_side = src.device.type == "cuda" or dst.device.type == "cuda"
        if has_cuda_side and transfer_embedding_ranges_direct is not None:
            transfer_embedding_ranges_direct(src, dst, src_starts, dst_starts, lengths)
            return

        for src_start, dst_start, valid_tokens in zip(src_starts, dst_starts, lengths):
            dst[dst_start : dst_start + valid_tokens].copy_(
                src[src_start : src_start + valid_tokens],
                non_blocking=True,
            )
```

test/registered/unit/mem_cache/test_embedding_cache_controller.py

覆盖 _build_storage_transfer_buffers 重命名后的行为，并新增碎片化 entry 的 H2D/D2H 拷贝计划期待值测试。

```
    def test_build_h2d_copy_plan_for_fragmented_entry(self):
        # 两条不连续 page run：PageRun(2, 1) 与 PageRun(7, 2)，共 5 个 token
        pool = _make_pool(num_pages=10, dim=4, page_size=2)
        entry = EmbeddingCacheEntry(
            hash="h",
```

```
            modality=Modality.IMAGE,
            num_tokens=5,
            dim=4,
            page_runs=[PageRun(2, 1), PageRun(7, 2)],
            state=EntryState.READY,
        )

        # H2D: pool 侧为源，目标写到设备 tensor 的 offset=3 处
        plan = _build_host_device_transfer_plan(
            entry, pool, src_is_pool=True, dst_token_offset=3
        )
        self.assertEqual(plan, ([4, 14], [3, 5], [2, 3]))


    def test_build_d2h_copy_plan_for_fragmented_entry(self):
        pool = _make_pool(num_pages=10, dim=4, page_size=2)
        entry = EmbeddingCacheEntry(
            hash="h",
            modality=Modality.IMAGE,
            num_tokens=5,
            dim=4,
            page_runs=[PageRun(2, 1), PageRun(7, 2)],
            state=EntryState.READY,
        )

        # D2H: pool 侧为目标，源是设备侧的连续 5 个 token
        plan = _build_host_device_transfer_plan(entry, pool, src_is_pool=False)
        self.assertEqual(plan, ([0, 2], [4, 14], [2, 3]))
```

python/sglang/kernels/aot/tests/test_kvcacheio.py

新增 CUDA 正确性测试：非默认流上验证 H2D/D2H 批量拷贝与 Tensor.copy_ 参考一致，覆盖碎片范围与不完整尾段。

```
# 参考实现：逐范围 Tensor.copy_（非阻塞），作为批量算子的正确性基准
def ref_copy_embedding_ranges(src, dst, src_starts, dst_starts, lengths):
    for src_start, dst_start, length in zip(src_starts, dst_starts, lengths):
        dst[dst_start : dst_start + length].copy_(
            src[src_start : src_start + length], non_blocking=True
        )


@pytest.mark.skipif(not torch.cuda.is_available(), reason="CUDA is required")
@pytest.mark.skipif(is_hip(), reason="This test covers the CUDA batch-copy op")
@pytest.mark.parametrize("direction", ["h2d", "d2h"])
def test_transfer_embedding_ranges_direct(direction: str):
    # 用非标准形状验证指针换算：37 维 embedding、page_size 4，最后一个范围只拷 2 行
    dtype = torch.bfloat16
    embedding_dim = 37
    page_size = 4
```

```
    fragmented_starts = [1, 11, 23]
    contiguous_starts = [2, 6, 10]
    lengths = [page_size, page_size, 2]
    host_rows = 28
    device_rows = 16

    host_values = torch.arange(host_rows * embedding_dim, dtype=torch.float32).reshape(
        host_rows, embedding_dim
    )
    device_values = torch.arange(
        device_rows * embedding_dim, dtype=torch.float32
    ).reshape(device_rows, embedding_dim)

    # H2D：源在 pinned host 的碎片位置，目标在 device 连续位置；D2H 反之
    if direction == "h2d":
        src = host_values.to(dtype).pin_memory()
        direct_dst = torch.full(
            (device_rows, embedding_dim), -1, dtype=dtype, device="cuda"
        )
        reference_dst = torch.full_like(direct_dst, -1)
        src_starts, dst_starts = fragmented_starts, contiguous_starts
    else:
        src = device_values.to(dtype).to("cuda")
        direct_dst = torch.full(
            (host_rows, embedding_dim), -1, dtype=dtype, pin_memory=True
        )
        reference_dst = torch.full(
            (host_rows, embedding_dim), -1, dtype=dtype, pin_memory=True
        )
        src_starts, dst_starts = contiguous_starts, fragmented_starts

    torch.cuda.synchronize()
    copy_stream = torch.cuda.Stream()
    # 必须跑在非默认流上，验证 stream 接线正确性
    assert copy_stream.cuda_stream != torch.cuda.default_stream().cuda_stream
    with torch.cuda.stream(copy_stream):
        ref_copy_embedding_ranges(src, reference_dst, src_starts, dst_starts, lengths)
        transfer_embedding_ranges_direct(
            src, direct_dst, src_starts, dst_starts, lengths
        )
        completion_event = torch.cuda.Event()
        completion_event.record(copy_stream)

    completion_event.synchronize()
    torch.testing.assert_close(direct_dst, reference_dst)
```

评论区精华

- BBuf 在 test_embedding_cache_controller.py:422 指出：「The new tests only validate the generated range lists and never invoke transfer_embedding_ranges_direct... A bug here could silently corrupt multimodal embeddings rather than fail immediately.」作者回复「Agreed. I'll add CUDA correctness tests to test_kvcacheio.py.」，并最终补齐非默认流、两方向、碎片的 CUDA 对照测试。
- BBuf 针对 H2D 源访问顺序建议：「could we benchmark cudaMemcpySrcAccessOrderAny here and keep cudaMemcpySrcAccessOrderStream for D2H? … vLLM adopted the same directional split in vLLM PR #39306.」作者在 H20/CUDA 13.0 实测两种顺序在 8-512 MiB 共 8 组下吞吐完全一致，并指出既有 KV 批拷路径双方向均用 Stream 顺序，最终保留 cudaMemcpySrcAccessOrderStream。
- gemini-code-assist[bot] 的两条 medium 建议在合入代码中未体现：一是在 Python 端检查 src.is_contiguous()/dst.is_contiguous() 以便安全回退到 copy_ 慢路径；二是把 attrs_idxs 从 std::vector(1, 0) 改为栈上 std::array 以规避热路径堆分配。
- BBuf 的总评同时要求补 benchmark 复现条件与 MM global-cache EPD 正确性测试，作者均在 PR body 中补全，并明确说明手动 EPD 验证「For this manual run, I temporarily bypassed the scheduler's batch_encode path... No scheduler change is included in this PR.」。
- CUDA 正确性测试覆盖不足 (testing): 作者同意并在 test_kvcacheio.py 补充 test_transfer_embedding_ranges_direct，覆盖非默认流、双方向、碎片范围与不完整尾段，对照 copy_ 参考。
- H2D 用 SrcAccessOrderAny 还是 Stream (performance): 作者在 H20/CUDA 13.0 实测 8-512 MiB 共 8 组数据两种顺序吞吐一致，并指出既有 KV 批拷路径也统一用 Stream 顺序；最终保留 cudaMemcpySrcAccessOrderStream，BBuf 表示非阻塞建议。
- Python 端连续性检查与 std::array 优化 (correctness): 合入代码未采纳这两条建议：Python 端仅检查设备侧存在性与算子可用性，attrs_idxs 仍为 std::vector(1, 0)。
- benchmark 可复现性与 EPD 端到端验证 (testing): 作者在 PR body 补齐完整 benchmark 脚本、CUDA 正确性测试与手动 EPD 双请求验证（第二轮 Local Hits=1），并说明手动验证临时绕过 scheduler batch_encode 路径。

风险与影响

- 风险：连续性、dtype、维度等校验位于 C++ TORCH_CHECK，Python 端 copy_embedding_page_runs 未前置检查 is_contiguous()；若未来调用方传入非连续张量，会直接抛 C++ 异常而非回退到 copy 循环（当前 pool.tensor 与 dst_tensor 均为连续，实际触发风险低）。CUDA 12.8/13.0 ABI 分派依赖运行时版本判断与 cudaMemcpyBatchAsync 符号及驱动版本探测；若出现新的 API 变体或驱动返回未知错误，将进入 TORCH_CHECK 直接失败而非回退，需要后续版本跟进。该算子仅在 CUDA 路径生效，HIP/MUSA 平台会走逐条 cudaMemcpyAsync 回退，AMD 等平台不获得收益但正确性不受影响；test_kvcacheio.py 对 HIP 显式 skip。端到端验证是手动临时绕过 scheduler batch_encode 完成的，未纳入自动化测试与调度器改动，真实调度路径下的 embedding cache 行为覆盖仍有限。thread_local std::vector 与 attrs_idxs 的堆分配在热路径上每次调用仍有一次小分配（bot 已指出，未采纳），相对拷本身体量占比很小。

- 影响：功能影响：启用多模态全局 embedding cache（EPD encoder-decoder + Mooncake）的用户是主要受益者；H20 上碎片化 H2D 吞吐提升约 42-43%、D2H 提升 28-48%，缓存越碎片化收益越明显；单范围与连续布局拷贝基本无变化。架构影响：build_transfer_buffers 更名为 build_storage_transfer_buffers，属于内部符号重命名，仓库内测试同步更新；存储路径与 host-device 路径的构建逻辑分离后更易扩展。兼容性：三级回退（批量 API → 逐条 cudaMemcpyAsync → Python copy 循环）保证旧环境与非 CUDA 平台行为不变；无配置或 schema 变更。
- 风险标记：CUDA 12.8/13.0 ABI 版本分支，非连续张量缺 Python 端回退，HIP 平台仅回退路径，端到端 EPD 测试未自动化

关联脉络

- PR #34121 [Diffusion] Fix cache-first fast path accepting a metadata-only snapshot: 同属多模态缓存正确性修复（cache-first 快速路径与快照语义），与本 PR 在多模态缓存命中与加载路径上领域相邻，但无文件交集。
- PR #34823 skip oow slot freeing under eagle: 同属 mem_cache 家族（radix cache 槽位释放）的碎片化 / 缓存一致性修复，与本 PR 都关注缓存碎片场景下的内存管理正确性，但属于 KV cache 与 speculative decoding 领域。