PR #37098 完整报告

sgl-project/sglang

[Unified Cache Linker][2/N]: Add device pool assembly for external linkers

合并时间：2026-08-30 23:54

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/37098

执行摘要

- 一句话：新增外部链接器设备池组装，支持 DSA 与 DeepSeek V4
- 推荐动作：值得精读。该 PR 是统一缓存外部链接器（HiCache）架构的关键一环，DevicePoolEntry 的元数据计算与 DevicePoolGroup.resolve_transfers 的展开语义是后续所有 linker 后端共用的基础。建议通读三个文件及配套测试，重点理解 packed/ 非 packed 布局、rows_are_pages 与稀疏 layer_mapping 的组合，以及 transfer 展开时 KV 强制 ALL_PAGES 的设计取舍。

功能与动机

该 PR 是外部链接器系列的第二部分，从 #35687 拆分。PR body 明确目标：为直接外部链接器增加可复用的设备池视图与传输解析，并先为 DSA 和 DeepSeek V4 落地设备池组装。背后的动机是让统一缓存（HiCache）支持外部链接器直接读写设备侧 KV 池，无需再走 host 侧中转，从而为后续后端无关 linker core（3/N）提供设备侧组装基础。

实现拆解

1. 新增 python/sglang/srt/mem_cache/hybrid_cache/linker_pool_assembler.py（380 行），定义 DevicePoolEntry 零拷贝视图：在 __init__ 中按组件计算每个 buffer 的 (data_ptr, row_stride, size) 三要素，支持 packed/ 非 packed 布局与稀疏 layer_mapping；_rows 做页对齐、页内连续与行范围校验，get_page_buffer_meta / get_prepared_layer_range_meta 分别返回整池与单层的指针 / 大小 / 偏移元数据。
2. 新增 DevicePoolGroup：把共享同一逻辑 linker 层范围的物理池聚合成组，resolve_transfers 将一个逻辑 KV transfer 按 sources 映射展开为多个物理池 transfer，并对索引做 translate_indices、对 KV 主池强制 PoolHitPolicy.ALL_PAGES；partial 覆盖与缺失 KV 默认拒绝，需显式 allow_partial / allow_missing_kv 才放行。
3. DeepSeek V4 组装：_build_deepseek_v4_device_pool_group 将 SWA、C4、C4_INDEXER、C128、C4_STATE、C4_INDEXER_STATE 六个实体装成 DevicePoolGroup；_deepseek_v4_state_views 把压缩状态池按 ring 大小切齐后重解释为 uint8 视图；同时拒绝 unified-KV 与 HiSparse，并要求 swa_page_size 与 tree page size 一致。
4. python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py 重构：抽取出 _DeepSeekV4LayerMappings NamedTuple 与 _resolve_deepseek_v4_layer_mappings，build_deepseek_v4_hicache_stack 增加可选参数 layer_mappings 复用解析结果；

StackStrategy 基类新增默认抛错的 build_direct_linker_pool_group，由 _DeepSeekV4Strategy 与 _DSAStrategy 各自实现，避免在基类堆叠分支。

5. 测试配套：新增 test/registered/unit/mem_cache/test_linker_pool_assembler.py（291 行，注册 base-a-test-cpu），覆盖稀疏多层范围元数据、非法页 / 空 buffer 拒绝、transfer 展开与索引翻译、partial sidecar 显式 opt-in、DeepSeek V4 稀疏 sidecar 映射、DSA 组装、不支持策略报错。CI 上 radix_cache/unified_radix_tree 组与该测试文件均 rerun 通过，AMD ROCm 7.2 工作流失败但未见说明。

关键文件：

- python/sglang/srt/mem_cache/hybrid_cache/linker_pool_assembler.py（模块 缓存池组装；类别 source；类型 core-logic；符号 DevicePoolEntry, DevicePoolGroup, resolve_hybrid_device_pool_group, _build_deepseek_v4_device_pool_group）：新增 380 行的核心模块，定义 DevicePoolEntry 零拷贝设备池视图、DevicePoolGroup transfer 展开、DeepSeek V4 与 DSA 设备池组装入口，是整套外部链接器系列的设备侧基础。
- python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py（模块 缓存池组装；类别 source；类型 refactor；符号 _DeepSeekV4LayerMappings, _resolve_deepseek_v4_layer_mappings, build_direct_linker_pool_group, build_deepseek_v4_hicache_stack）：抽取 DeepSeek V4 层映射解析逻辑（_resolve_deepseek_v4_layer_mappings），并给 StackStrategy 增加 build_direct_linker_pool_group 入口，让 DSA/DeepSeek V4 策略各自实现而不污染基类。
- test/registered/unit/mem_cache/test_linker_pool_assembler.py（模块 池组装测试；类别 test；类型 test-coverage；符号 TestDevicePoolEntry, test_sparse_multi_component_layer_ranges, test_rejects_invalid_pages_and_empty_buffers, TestDevicePoolGroup）：291 行单元测试，覆盖 DevicePoolEntry 稀疏多层范围、非法页拒绝、transfer 展开、partial 显式 opt-in、DeepSeek V4 稀疏 sidecar 与 DSA 组装，是整个组装层正确性的主要保障。

关键符号：DevicePoolEntry.init, DevicePoolEntry._rows, DevicePoolEntry.get_page_buffer_meta, DevicePoolEntry.get_prepared_layer_range_meta, DevicePoolGroup.resolve_transfers, resolve_hybrid_device_pool_group, _resolve_deepseek_v4_layer_mappings, _build_deepseek_v4_device_pool_group, _build_dsa_device_pool_group, build_direct_linker_pool_group

关键源码片段

python/sglang/srt/mem_cache/hybrid_cache/linker_pool_assembler.py

新增 380 行的核心模块，定义 DevicePoolEntry 零拷贝设备池视图、DevicePoolGroup transfer 展开、DeepSeek V4 与 DSA 设备池组装入口，是整套外部链接器系列的设备侧基础。

```
class DevicePoolEntry:
    """Zero-copy linker view over one physical device pool."""

    def _rows(self, indices: torch.Tensor) -> list[int]:
        # 将设备索引搬回 CPU 并展平，统一用 int64 做页对齐校验
```

```
    slots = indices.detach().to(device="cpu", dtype=torch.int64).flatten()
    if slots.numel() % self.page_size:
        raise ValueError(
            f"Pool {self.name} got {slots.numel()} indices, expected a "
            f"multiple of page_size={self.page_size}."
        )
    if not slots.numel():
        return []

    # 要求索引按 page_size 对齐且页内连续，否则无法映射为整页
    pages = slots.reshape(-1, self.page_size)
    starts = pages[:, 0]
    if torch.any(starts.remainder(self.page_size)) or not torch.equal(
        pages, starts[:, None] + self._page_offsets
    ):
        raise ValueError(f"Pool {self.name} requires aligned contiguous pages.")

    # rows_are_pages 时一行即一页；否则一行等于一个 page_size 的段
    rows = (
        starts.div(self.page_size, rounding_mode="floor")
        if self._row_span == 1
        else starts
    )
    first_row = int(rows.min())
    last_row = int(rows.max()) + self._row_span
    if first_row < 0 or last_row > self._row_count:
        raise ValueError(
            f"Pool {self.name} row range [{first_row}, {last_row}) exceeds "
            f"buffer shapes {[tuple(buffer.shape) for buffer in self.kv_buffer]}."
        )
    return rows.tolist()

def get_prepared_layer_range_meta(self, locations: list[int], layer: int):
    # 按全局层号查局部 buffer 下标；未映射的层返回 None，由调用方跳过
    buffer_index = self.layer_mapping.get(layer)
    if buffer_index is None:
        return None

    items = []
    for component, offsets in zip(self.buffer_meta, self._component_offsets):
        base_ptr, row_stride, size = component[buffer_index]
        items.append((base_ptr, row_stride, size, offsets[buffer_index]))

    ptrs, sizes, offsets = [], [], []
    for row in locations:
        row_ptrs = [
            base_ptr + row * row_stride for base_ptr, row_stride, _, _ in items
        ]
        row_sizes = [size for _, _, size, _ in items]
```

```
            row_offsets = [offset for _, _, _, offset in items]
            if self.packed:
                # packed 布局下多个缓冲区按行打包，需要逐行给出指针 / 大小 / 偏移
                ptrs.append(row_ptrs)
                sizes.append(row_sizes)
                offsets.append(row_offsets)
            else:
                # 非 packed 布局展开为独立项，供 linker 逐行处理
                ptrs.extend([[value] for value in row_ptrs])
                sizes.extend([[value] for value in row_sizes])
                offsets.extend([[value] for value in row_offsets])
        return ptrs, sizes, offsets
```

python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py

抽取 DeepSeek V4 层映射解析逻辑（_resolve_deepseek_v4_layer_mappings），并给 StackStrategy 增加 build_direct_linker_pool_group 入口，让 DSA/DeepSeek V4 策略各自实现而不污染基类。

```
class _DeepSeekV4LayerMappings(NamedTuple):
    transfer_layer_num: int
    full: dict[int, int]
    swa: dict[int, int]
    c4: dict[int, int]
    c128: dict[int, int]
    c4_state: dict[int, int]
    c4_state_global_layers: list[int]


def _resolve_deepseek_v4_layer_mappings(
    kvcache: Any,
) -> _DeepSeekV4LayerMappings:
    # transfer 只覆盖本 PP 分区的局部层区间
    transfer_layer_num = kvcache.end_layer - kvcache.start_layer
    full = {layer: layer for layer in range(transfer_layer_num)}
    # unified KV 下 SWA 环住在统一池里，不再有独立 SWA 映射
    swa = {} if getattr(kvcache, "_unified_kv", False) else full.copy()

    c4, c128, c4_state_global_layers = {}, {}, []
    # 按压缩比把局部层映射到压缩层 id，并记录 C4 状态池对应的全局层
    for local_layer, item in enumerate(
        kvcache.layer_mapping[kvcache.start_layer : kvcache.end_layer]
    ):
        if item.compress_ratio == 4:
            c4[local_layer] = item.compress_layer_id
            c4_state_global_layers.append(kvcache.start_layer + local_layer)
        elif item.compress_ratio == 128:
            c128[local_layer] = item.compress_layer_id

    return _DeepSeekV4LayerMappings(
```

```
        transfer_layer_num=transfer_layer_num,
        full=full,
        swa=swa,
        c4=c4,
        c128=c128,
        # c4_state 用枚举序作为状态池下标，与全局层列表一一对应
        c4_state={layer: index for index, layer in enumerate(c4)},
        c4_state_global_layers=c4_state_global_layers,
    )
```

评论区精华

本 PR 没有实质 review 评论线程。可观察到的交互是作者发起两次 CI 重跑：/rerun-group radix_cache/unified_radix_tree（4-gpu-h100、4-gpu-b200、8-gpu-h200 全部通过）与 /rerun-test test/registered/unit/mem_cache/test_linker_pool_assembler.py（ubuntu-latest 通过）。AMD ROCm 7.2 的 PR test 显示 X，但没有对应评论解释，属于未闭环的 CI 疑问。

- 暂无高价值评论线程

风险与影响

- 风险：
  1. 新模块未接入运行时：本 PR 只是组装层，调用点来自后续 PR；若后续接入时对 DevicePoolEntry 的假设（packed、rows_are_pages、layer_mapping 稀疏性）理解不一致，可能返工。
  2. 严格校验：_rows 要求索引是 page_size 的倍数、页对齐且页内连续，测试覆盖了错误路径；但若未来出现非连续页组（碎片化分页）会直接抛 ValueError，调用方需保证输入形态。
  3. _deepseek_v4_state_views 用 view(torch.uint8) 重解释 kv_score 缓冲区，依赖张量连续与内存对齐；非 contiguous 张量会导致 view 失败或数据错位。
  4. resolve_transfers 对 KV 主池强制 PoolHitPolicy.ALL_PAGES，会改变原有 TRAILING_PAGES 语义；partial 转移默认被拒，每个调用点需显式打开 allow_partial。
  5. 零拷贝视图只持有 device_pool 引用，不管理生命周期；池被释放或重建后，缓存的 data_ptr 元数据会悬挂。
  6. AMD ROCm 7.2 CI 失败未解释，需确认是否与本 PR 相关。
  7. 测试基于 CPU 缓冲计算指针，不能覆盖跨设备寻址、CUDA 分配器导致的地址漂移等真实场景。 - 影响：对用户无直接行为变化，新的 resolve_hybrid_device_pool_group 尚未被调度路径调用。系统侧为外部链接器直接读写设备 KV 池建立了数据契约（指针 / 大小 / 偏移三要素、层映射、页大小一致性），DeepSeek V4 侧要求 swa_page_size 与 tree page size 一致，约束后续 linker 的配置面。团队侧与 3/N（#37151 backend-independent linker core）形成接力，2/N 提供设备池组装，3/N 提供后端无关 core，需要同步 review 理解整体设计。重构受益方面，_resolve_deepseek_v4_layer_mappings 消除了 HiCache 栈内重复的层映射计算，后续

新增压缩层类型只需改一处。 - 风险标记：新模块未接入运行时，严格索引校验依赖调用方，state 视图依赖张量连续性，KV 强制 ALL_PAGES 语义变更，AMD CI 失败未解释

关联脉络

- PR #35687 [Unified Cache Linker] 原始大 PR: 本 PR 明确说明是从 #35687 拆分出来的第二个 PR，是该系列的拆分来源。
- PR #37151 [Unified Cache Linker][3/N]: Add backend-independent linker core: 同系列下一步：在后端无关 linker core 中消费本 PR 提供的设备池组装结果，两者构成完整外部链接器链路。
- PR #35245 refactor(unified-memory): translate the KV write location once, at ForwardBatch construction: 统一内存 KV 索引翻译思路与 DevicePoolEntry.translate_indices 同源，反映 KV 索引翻译统一化的演进方向。
- PR #34602 feat(unified-memory): dense KV views for uniform-row MHA/SWA models: 统一池 dense 视图设计与本 PR 的设备池零拷贝视图设计相关，同属内存视图抽象演进。