PR #24691 完整报告

sgl-project/sglang

[UnifiedTree]: Support HiCache For DeepSeek_V4

合并时间：2026-05-15 11:20

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/24691

执行摘要

- 一句话：DeepSeek V4 集成 HiCache，引入 Sidecar 池复用索引。
- 推荐动作：建议精读核心文件：memory_pool_host.py 中的 LogicalHostPool 设计展示了如何以纯逻辑池作为锚点，hybrid_cache_controller.py 中的 sidecar 解析演示了懒绑定索引的使用模式，适合作为缓存层次化扩展的参考。注意当前限制（仅 KV 源、无 kernel 后端）并关注后续改进。

功能与动机

扩展 UnifiedTree 的 HiCache 支持到 DeepSeek V4 模型，实现 CPU 卸载以减少 GPU 显存占用，是统一混合缓存重构路线图（#20415）的一部分。PR body 指出这是对 SWA HiCache (#23391) 的跟进，利用 UnifiedTree 的 SWA HiCache 能力和 Shadow Radix 机制。

实现拆解

1. 引入 V4 专用主机池类：在 memory_pool_host.py 中新增 LogicalHostPool（纯逻辑锚池，管理页对齐槽位索引）和 DeepSeekV4PagedHostPool/DeepSeekV4StateHostPool（继承 HostKVCache，管理压缩 KV、索引器和状态数据的 CPU 侧缓存）。
2. 构建 V4 HiCache 堆栈：在 hybrid_pool_assembler.py 中新增 _deepseek_v4_num_host_pages、build_deepseek_v4_hicache_stack、build_shared_anchor_stack、build_anchor_sidecar_stack 等函数。根据设备池容量和 --hicache-ratio 计算主机页数，创建锚池和侧车池的 PoolEntry，组装成 HostPoolGroup，并实例化 HybridCacheController。
3. 改进混合缓存控制器：在 hybrid_cache_controller.py 中，将 _resolve_shared_pool_transfers 替换为 _resolve_sidecar_derived_pool_transfers，支持通过 indices_from_pool 字段从源池懒继承索引。merge_pool_transfers 的键从 PoolName 扩展为 (PoolName, Optional[PoolName]) 元组以区分传输来源；新增 rollback_allocated 用于原子回滚。
4. 集成到 UnifiedRadixCache：在 unified_radix_cache.py 中，以 sidecar_pool_specs 列表替代旧的 hicache_anchor_kv_shared_indices_pools，新增 register_sidecar_pool 和 _build_sidecar_transfers 方法。在 write_backup 和 load 操作中，根据 SidecarPoolSpec 生成对应的 PoolTransfer 对象，由控制器解析执行。

5. 配套测试与配置调整：新增 test/registered/radix_cache/test_unified_radix_hicache_kl.py，包含 Mamba 和 DeepSeek V4 Flash 的 HiCache KL 散度测试；修改 test_unified_radix_cache_kl.py 移除旧的 Mamba HiCache 测试，并调整 kl_multiturn_utils.py 等工具函数。

关键文件：

- python/sglang/srt/mem_cache/memory_pool_host.py（模块 主机池；类别 source；类型 core-logic；符号 LogicalHostPool, init, clear, available_size）：引入 LogicalHostPool 和 DeepSeekV4PagedHostPool 等新主机池类，构成 V4 HiCache 的核心数据结构。
- python/sglang/srt/mem_cache/hybrid_cache/hybrid_pool_assembler.py（模块 池组装；类别 source；类型 dependency-wiring；符号 _deepseek_v4_num_host_pages, build_deepseek_v4_hicache_stack, build_shared_anchor_stack, build_anchor_sidecar_stack）：核心装配逻辑：根据 V4 模型层映射和压缩比构建主机池组与控制器，定义 sidecar 池的派生关系。
- python/sglang/srt/mem_cache/hybrid_cache/hybrid_cache_controller.py（模块 缓存控制；类别 source；类型 entrypoint；符号 merge_ops, _resolve_shared_pool_transfers, _resolve_sidecar_derived_pool_transfers, rollback_allocated）：控制器核心：修改 merge_pool_transfers 分组键，新增 _resolve_sidecar_derived_pool_transfers 实现 sidecar 懒绑定，替换旧的共享池逻辑。
- python/sglang/srt/mem_cache/unified_radix_cache.py（模块 统一缓存；类别 source；类型 dependency-wiring；符号 register_hicache_anchor_kv_shared_indices_pool, register_sidecar_pool, _build_sidecar_transfers）：UnifiedRadixCache 集成 sidecar 池：以 sidecar_pool_specs 替代旧列表，新增注册和构建传输方法。
- test/registered/radix_cache/test_unified_radix_hicache_kl.py（模块 测试；类别 test；类型 test-coverage；符号 TestUnifiedMambaHiCache, setUpClass, tearDownClass, _assert_dsv4_decode_cached_tokens）：新增 HiCache KL 散度测试，覆盖 DeepSeek V4 Flash 和 Mamba 模型的多轮缓存正确性。
- python/sglang/srt/mem_cache/hicache_storage.py（模块 存储层；类别 source；类型 core-logic；符号 SidecarPoolSpec）：新增 V4 池名称枚举和 SidecarPoolSpec 数据类，定义 sidecar 池配置结构。

关键符号：LogicalHostPool, DeepSeekV4PagedHostPool, DeepSeekV4StateHostPool, build_deepseek_v4_hicache_stack, build_shared_anchor_stack, build_anchor_sidecar_stack, _deepseek_v4_num_host_pages, _resolve_sidecar_derived_pool_transfers, merge_pool_transfers, register_sidecar_pool, _build_sidecar_transfers, TestUnifiedDeepSeekV4FlashHiCache

关键源码片段

python/sglang/srt/mem_cache/memory_pool_host.py

引入 LogicalHostPool 和 DeepSeekV4PagedHostPool 等新主机池类，构成 V4 HiCache 的核心数据结构。

```
class LogicalHostPool:
```

```
"""Pure-logical anchor pool for V4 HiCache.

管理页对齐令牌槽位，不持有 KV 张量。
压缩侧车池使用此池的完整索引作为稳定页锚点。
"""

def __init__(self, size: int, page_size: int):
    if size % page_size != 0:
        raise ValueError(
            "LogicalHostPool size must be page-aligned, "
            f"got size={size}, page_size={page_size}"
        )
    self.size = size
    self.page_size = page_size
    self.device = "cpu"
    self.layout = "layer_first"
    self.dtype = torch.uint8
    self.layer_num = 0
    self.start_layer = 0
    self.end_layer = 0
    self.kv_buffer = None
    self.size_per_token = 0
    self.allocator = None
    self.lock = threading.RLock()
    self.clear()

@synchronized
def clear(self):
    # 重置空闲槽位列表为全部索引
    self.free_slots = torch.arange(self.size, dtype=torch.int64)

def available_size(self):
    return len(self.free_slots)

@synchronized
def alloc(self, need_size: int) -> Optional[torch.Tensor]:
    # 只允许页对齐分配
    if need_size % self.page_size != 0:
        raise ValueError(
            "LogicalHostPool allocation must be page-aligned, "
            f"got need_size={need_size}, page_size={self.page_size}"
        )
    if need_size > self.available_size():
        return None
    select_index = self.free_slots[:need_size]
    self.free_slots = self.free_slots[need_size:]
    return select_index

@synchronized
```

```
    def free(self, indices: torch.Tensor) -> int:
        if len(indices) % self.page_size != 0:
            raise ValueError(
                "LogicalHostPool free must be page-aligned, "
                f"got len(indices)={len(indices)}, page_size={self.page_size}"
            )
        # 归还索引到空闲列表
        self.free_slots = torch.cat(
            [self.free_slots, indices.to(dtype=torch.int64, device="cpu").flatten()]
        )
        return len(indices)

    # 以下方法为空操作：LogicalHostPool 不持有实际 KV 数据
    def backup_from_device_all_layer(self, device_pool, host_indices, device_indices, io_backend):
        pass

    def load_to_device_per_layer(self, device_pool, host_indices, device_indices, layer_id, io_
    backend):
        pass

    def get_data_page(self, index, flat=True):
        return torch.empty(0, dtype=torch.uint8)

    def get_dummy_flat_data_page(self):
        return torch.empty(0, dtype=torch.uint8)

    def set_from_flat_data_page(self, index, data_page):
        pass

    def get_page_buffer_meta(self, indices):
        return None

    def get_ksize_per_token(self):
        return 0
```

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

控制器核心：修改 merge_pool_transfers 分组键，新增 _resolve_sidecar_derived_pool_transfers 实现 sidecar 懒绑定，替换旧的共享池逻辑。

```
  def _resolve_sidecar_derived_pool_transfers(self, operation):
    """将 sidecar 传输的索引懒绑定为操作的主 KV 索引。"""
    for transfer in operation.pool_transfers:
        if transfer.indices_from_pool is None:
            # 非派生池，跳过（如独立分配的 Mamba/SWA）
            continue
        if transfer.indices_from_pool != PoolName.KV:
            # TODO(hzh): 支持从 SWA 等其他源池派生
            raise AssertionError(
                "Storage sidecar derived pool currently only supports KV-shared "
```

```
                f"indices, got {transfer.name} from {transfer.indices_from_pool}."
            )
        # 懒绑定：用操作的主 KV 索引作为侧车池索引
        transfer.host_indices = operation.host_indices
        if transfer.keys is not None and operation.keys is not None:
            transfer.keys = operation.keys
        # 如果 sidecar 传输本身带有 device_indices，则保留；否则从操作继承
        if transfer.device_indices is None:
            transfer.device_indices = operation.device_indices

@staticmethod
def merge_pool_transfers(ops: List[CacheOperation]) -> Optional[list[PoolTransfer]]:
    # 旧版：按 PoolName 分组（单个键）
    # 新版：按 (PoolName, Optional[PoolName]) 分组，以区分来自不同源池的传输（如 DEEPSEEK_
    V4_C4 从 KV 还是 SWA 派生）
    grouped: dict[tuple[PoolName, Optional[PoolName]], list[PoolTransfer]] = {}
    for op in ops:
        for t in op.pool_transfers or []:
            grouped.setdefault((t.name, t.indices_from_pool), []).append(t)
    # ...
    return [
        PoolTransfer(
            name=ts[0].name,
            host_indices=cat_or_none(...,)) ,
            # 保留源池信息用于后续解析
            hit_policy=ts[0].hit_policy,
            indices_from_pool=ts[0].indices_from_pool,
        )
        for ts in grouped.values()
    ]
```

评论区精华

Review 中讨论了以下核心问题：

- Sidecar 解析仅支持 KV 来源：gemini-code-assist[bot] 指出 _resolve_sidecar_derived_pool_transfers 只处理 PoolName.KV，但状态池从 SWA 派生，会导致断言失败。作者确认并在代码中添加 TODO，承诺后续支持其他来源。
- PoolName 枚举重复：DEEPSEEK_V4_INDEXER 与 DEEPSEEK_V4_C4_INDEXER 值相同，ispobock 指正。作者同意并计划在后续 PR 中规范化命名。
- PoolName 通用性建议：ispobock 建议将 V4 特定命名改为 COMPRESSED_KV 等通用类型，作者添加 TODO 记录。
- SidecarPoolSpec 与 PoolTransfer 字段区别：ispobock 质疑两者 indices_from_pool 重复，作者解释 SidecarPoolSpec 是配置声明，PoolTransfer 的字段是运行时懒绑定必需，不可移除。
    - Sidecar 解析仅支持 KV 来源会导致状态池崩溃 (correctness)：作者添加 TODO，计划后续支持其他源池；当前使用 assert 阻止非预期路径，生产环境中若配置使用会崩溃。

- PoolName 枚举重复：DEEPSEEK_V4_INDEXER (style): 作者同意，将在后续 PR 中规范化命名。
- PoolName 应改为通用类型名而非 V4 特定 (design): 作者同意并添加 TODO 注释，计划后续重构。
- SidecarPoolSpec 与 PoolTransfer.indices_from_pool 字段区别 (design): 作者解释 SidecarPoolSpec 是树层的配置声明（不可变），PoolTransfer.indices_from_pool 是运行时在控制器中懒绑定必需的，两者角色不同，不能移除。

风险与影响

- 风险：
  1. 核心路径变更风险：sidecar 解析逻辑当前仅支持从 KV 池继承索引，任何从其他源池（如 SWA）导出的状态池在备份 / 加载时会触发断言失败，全功能支持需后续 PR 完善。
  2. 内存管理风险：新增的 DeepSeekV4PagedHostPool 是复杂的 HostKVCache 子类，页分配 / 释放逻辑可能存在并发问题或边界错误（如非页对齐请求虽已校验，但动态路径仍可能因 off-by-one 导致崩溃）。
  3. 性能风险：Issue 评论中用户报告在 16k prefill 工作负载下吞吐下降约 20%，Nsight 显示 H2D memcpy 占比显著，当前仅支持 direct IO 后端，kernel 后端（#25282）尚未就绪。
  4. 测试覆盖不足：当前仅包含 Mamba 和 DeepSeek V4 Flash 的 KL 散度测试，缺少对状态池、长序列、高并发等场景的覆盖，且部分测试被标记为 skip（如 test_multiturn_logprobs_match）。
- 影响：
  - 用户：DeepSeek V4 模型启用 --enable-hierarchical-cache 后可使用 HiCache 减少 GPU 显存占用（通过卸载到 CPU），但 initial 版本仅支持 direct 后端，性能收益受限。
  - 系统：引入新的主机池类型和 sidecar 池机制，内存管理复杂度增加；PoolTransfer 新增 indices_from_pool 字段影响所有 HiCache 传输路径；merge_pool_transfers 分组键变更可能影响缓存合并行为。
  - 团队：这是 UnifiedTree 重构的关键一步，后续 PR 将规范化 PoolName 并支持更多模型族的 sidecar 池，当前设计为可扩展性提供了范例。
  - 风险标记：Sidecar 解析仅支持 KV 源，缺少状态池覆盖，CPU-GPU 传输性能瓶颈，kernel 后端待支持

关联脉络

- PR #23391 [UnifiedTree]: Support HiCache For SWA: 本 PR 是 SWA HiCache 的 DeepSeek V4 扩展，复用其 Shadow Radix 机制。
- PR #21206 [UnifiedTree]: Support Unified HybridRadixTree V2: 提供了 UnifiedRadixCache 框架，本 PR 在此基础上集成 HiCache。
- PR #20415 [Roadmap] Unified Hybrid Radix Cache Refactor: 本 PR 是该路线图的 Stage 1 关键交付，实现 HiCache 对 DeepSeek V4 的支持。