PR #25824 完整报告

sgl-project/sglang

[Refactor] Encapsulate SWA loc translation inside SWAKVPool with per-batch cache invalidation

合并时间：2026-05-21 12:26

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25824

执行摘要

- 一句话：封装 SWA 位置转换到 SWAKVPool 并添加逐缓存失效
- 推荐动作：值得精读。设计上通过精心选择缓存键和失效策略在不需要外部协调的情况下实现了正确性。特别是 data_ptr 与 storage data_ptr 的区别、CUDA 图捕获中的缓存生命周期控制，是值得注意的设计细节。

功能与动机

此前调用方需在每个 forward pass 前通过 set_swa_loc 设置 SWA 位置，且 out_cache_loc_swa 作为 ForwardBatch 字段需在每个图运行器中切片和传递。此重构消除这一侧信道，将转换逻辑封装在 SWAKVPool 内，减少调用方耦合和潜在错误。

实现拆解

1. 基类与核心缓存：在 BaseSWAKVPool 中添加 no-op invalidate_loc_cache()，在 SWAKVPool 中实现基于 (data_ptr, numel) 的缓存，移除 set_swa_loc 和 swa_loc 属性。
2. 惰性翻译：修改 translate_loc_from_full_to_swa 以缓存键 (kv_indices.data_ptr(), kv_indices.numel())，仅在缓存未命中时执行 gather 操作。
3. 自动失效：在所有修改 full_to_swa_index_mapping 的方法（alloc, free_swa, clear, set_full_to_swa_mapping, register_mapping）中调用 invalidate_loc_cache()，确保映射变更后缓存立即失效。
4. CUDA 图安全：在所有 CUDA 图运行器的 run_once() 开头调用 invalidate_loc_cache()（包括 piecewise, breakable, 标准 cuda_graph_runner 以及 speculative runners），防止 warmup 缓存污染 capture 运行。
5. 数据契约清理：移除 ForwardBatch.out_cache_loc_swa 字段，以及所有相关的缓冲区创建和拷贝（DecodeInputBuffers, PrefillInputBuffers 等），并清理对应引用点（radix_attention, triton_backend, trtllm_mha_backend 等）。
6. 测试覆盖：新增 test/manual/core/test_swa_loc_translation_cache.py，覆盖缓存键正确性（data_ptr 区分偏移）、分配器变异失效，以及基类 no-op。

关键文件：

- test/manual/core/test_swa_loc_translation_cache.py（模块 测试；类别 test；类型 test-coverage；符号 _build_pool, TestCacheKeyDataPtr, test_same_offset_view_is_cache_hit, test_different_offset_view_is_cache_miss）：新增

单元测试，覆盖缓存键正确性、分配器变异失效、基类 no-op，是确保重构正确性的关键配套。

- python/sglang/srt/mem_cache/swa_memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 invalidate_loc_cache, set_swa_loc, translate_loc_from_full_to_swa）：核心逻辑修改：添加基于 (data_ptr, numel) 的缓存、移除 set_swa_loc、在 7 个分配器变异点插入 invalidate_loc_cache()。
- python/sglang/srt/model_executor/piecewise_cuda_graph_runner.py（模块 图运行器；类别 source；类型 data-contract）：移除 out_cache_loc_swa 数据契约，在 run_once() 中添加 invalidate_loc_cache() 确保 CUDA 图安全。
- python/sglang/srt/model_executor/cuda_graph_runner.py（模块 图运行器；类别 source；类型 data-contract）：移除 DecodeInputBuffers.out_cache_loc_swa 字段；移除 is_hybrid_swa 参数及相关拷贝逻辑；在 run_once() 中添加失效。
- python/sglang/srt/mem_cache/base_swa_memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 invalidate_loc_cache, set_swa_loc）：添加 no-op invalidate_loc_cache() 默认实现，移除抽象方法 set_swa_loc，为子类提供平滑过渡。
- python/sglang/srt/model_executor/forward_batch_info.py（模块 前向批；类别 source；类型 data-contract）：移除 ForwardBatch 的 out_cache_loc_swa 字段，消除预计算 SWA 位置的数据契约。

关键符号：invalidate_loc_cache, translate_loc_from_full_to_swa, set_kv_buffer, register_mapping, alloc, free_swa, clear, set_full_to_swa_mapping, _forward_raw, run_once, DecodeInputBuffers.create, DecodeInputBuffers.populate_from_forward_batch

关键源码片段

test/manual/core/test_swa_loc_translation_cache.py

新增单元测试，覆盖缓存键正确性、分配器变异失效、基类 no-op，是确保重构正确性的关键配套。

```
"""Manual tests for SWAKVPool.translate_loc_from_full_to_swa cache behavior.

These tests cover three properties introduced by PR #25824:
  1. Cache key uses data_ptr() — correctly distinguishes views at different
     offsets within the same storage (untyped_storage().data_ptr() would not).
  2. Allocator mutations invalidate the cache — alloc/free/clear/
     set_full_to_swa_mapping each call invalidate_loc_cache() so the next
     translation sees the fresh mapping.
  3. BaseSWAKVPool.invalidate_loc_cache is a no-op default — subclasses that
     don't cache (e.g. DSV4) can be called safely without AttributeError.
Run with:
    python -m pytest test/manual/core/test_swa_loc_translation_cache.py -v
"""

def _build_pool(kv_size=32, kv_size_swa=32, page_size=1):
    # ... 省略构建代码 ...
```

```
    return pool, allocator, device


class TestCacheKeyDataPtr(CustomTestCase):
    """Cache key uses data_ptr(), which encodes the storage offset."""

    def test_same_offset_view_is_cache_hit(self):
        """Two different Python objects pointing to the same base are a hit."""
        pool, allocator, device = _build_pool()
        loc = allocator.alloc(4)
        self.assertIsNotNone(loc)
        # 创建两个偏移量 0 的切片对象 — 相同 data_ptr, 相同 numel
        view_a = loc[:4]
        view_b = loc[:4]
        self.assertIsNot(view_a, view_b) # 不同 Python 对象
        self.assertEqual(view_a.data_ptr(), view_b.data_ptr())
        result_a = pool.translate_loc_from_full_to_swa(view_a)
        result_b = pool.translate_loc_from_full_to_swa(view_b)
        # 两者应返回同一张量（缓存命中）
        self.assertIs(result_a, result_b)

    def test_different_offset_view_is_cache_miss(self):
        """Views at different offsets produce different data_ptr → cache miss."""
        pool, allocator, device = _build_pool()
        loc = allocator.alloc(10)
        view_lo = loc[0:5]
        view_hi = loc[5:10]
        # 不同 data_ptr（不同存储偏移）
        self.assertNotEqual(view_lo.data_ptr(), view_hi.data_ptr())
        result_lo = pool.translate_loc_from_full_to_swa(view_lo)
        result_hi = pool.translate_loc_from_full_to_swa(view_hi)
        self.assertIsNot(result_lo, result_hi)
        self.assertFalse(torch.equal(result_lo, result_hi))

    def test_storage_base_ptr_would_collide(self):
        """Demonstrate that untyped_storage().data_ptr() WOULD collide."""
        t = torch.arange(20, device=get_device())
        a, b = t[0:10], t[5:15]
        # 相同 storage base — 旧键会冲突
        self.assertEqual(a.untyped_storage().data_ptr(), b.untyped_storage().data_ptr())
        # 但 data_ptr 不同 — 新键安全
        self.assertNotEqual(a.data_ptr(), b.data_ptr())
```

python/sglang/srt/mem_cache/swa_memory_pool.py

核心逻辑修改：添加基于 (data_ptr, numel) 的缓存、移除 set_swa_loc、在 7 个分配器变异点插入 invalidate_loc_cache()。

```
    def invalidate_loc_cache(self) -> None:
        # 将缓存标记为失效，下一次 translate 会重新计算
```

```
        self._cached_swa_loc = None
        self._cached_loc_key = None

    def translate_loc_from_full_to_swa(self, kv_indices: torch.Tensor) -> torch.Tensor:
        assert self.full_to_swa_index_mapping is not None
        # 使用 data_ptr()（而非 untyped_storage().data_ptr()）编码切片偏移，
        # 确保同一 storage 中不同位置的视图获得不同缓存键。
        # -1 在 kv_indices 中会映射到 -1（通过 mapping 末尾的 sentinel）。
        key = (kv_indices.data_ptr(), kv_indices.numel())
        if key != self._cached_loc_key:
            if self._cached_loc_key is not None:
                logger.warning(
                    "translate_loc_from_full_to_swa: loc tensor changed mid-forward "
                    "without invalidate_loc_cache() — possible missing call site"
                )
            self._cached_swa_loc = self.full_to_swa_index_mapping[kv_indices].to(
                torch.int32
            )
            self._cached_loc_key = key
        return self._cached_swa_loc

    def set_kv_buffer(self, layer, loc, skip_wait=False):
        layer_id = layer.layer_id
        layer_id_pool, is_swa_layer = self.layers_mapping[layer_id]
        if is_swa_layer:
            # 直接翻译，无需外部预置 swa_loc
            loc = self.translate_loc_from_full_to_swa(loc)
            self.swa_kv_pool.set_kv_buffer(None, loc, skip_wait)
        else:
            self.full_kv_pool.set_kv_buffer(None, loc, skip_wait)
```

评论区精华

该 PR 未有 review 评论，但从 commit 历史可见多次设计迭代：

- 最初使用 layer_id_pool==0 作为失效信令，后改为 loc.data_ptr() 键，避免层序耦合。
- 确认 untyped_storage().data_ptr() 会导致不同偏移的视图冲突，改用 data_ptr()。
- 添加对 HiCache 场景下 rebuild mapping 时的失效。
- 在 CUDA 图捕获路径中发现 warmup-1 缓存污染，强制在 run_once 中失效。 这些迭代体现了对正确性的谨慎态度。
- 暂无高价值评论线程

风险与影响

- 风险：主要风险是缓存失效遗漏：若某修改 full_to_swa_index_mapping 的函数忘记调用 invalidate_loc_cache()，将导致后续 translate 返回过时 SWA 索引，可能引发越界或数据损坏。当前通过代码审核和测试覆盖所有已知变异点，但未来新增变异点需同步添加。

CUDA 图模式下，若 run_once() 的失效点被移除或绕过，会导致捕获图不包含 gather 内核，进而回放时使用旧索引。移除 out_cache_loc_swa 字段可能影响第三方通过子类化 ForwardBatch 或自定义 CUDA 图运行器的扩展。

- 影响：对使用 SWA 的模型（如 DeepSeek V3/V4）的推理性能无直接影响（默认路径计算不变）；缓存机制在拥有多个 SWA 层时可减少重复 gather 开销（首次 SWA 层计算，后续缓存命中）。影响范围涉及 token_to_kv_pool、CUDA 图运行器、注意力后端等 18 个文件，均为内部数据契约调整，无公共 API 变化。用户无需更改配置或代码。
- 风险标记：缓存失效遗漏风险，CUDA 图缓存污染，跨模块耦合，新分配器方法需同步失效

关联脉络

- PR #24226 [BugFix] fix(hicache): fix two slot-reuse races in DecodeKVCacheOffloadManager: 该 PR 修复了 HiCache 中的 slot 重用竞态，与本 PR 的缓存失效逻辑有相同的正确性关注点，且本 PR 的 commit 历史中曾修复过 HiCache 场景下的映射重建后未失效的问题。