PR #27256 完整报告

sgl-project/sglang

[mem_cache][4/N] refactor: extract MambaTokenToKVPoolAllocator into allocator/

合并时间：2026-06-07 10:46

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/27256

执行摘要

- 一句话：提取 Mamba 插槽分配器到 allocator/ 子包
- 推荐动作：值得精读。展示了如何将内联分配逻辑提取为独立组件，以及重构中的设计选择（不继承 KV 分配器基类）。对于理解 SGLang 内存管理架构有帮助。

功能与动机

作为 mem_cache 重构 (Issue #25371) 的一部分，目标是将分配器与物理存储解耦，使代码模块化。本 PR 提取 Mamba 状态池的分配器，与之前对 KV 分配器的处理一致。

实现拆解

1. 创建分配器类：在 allocator/mamba.py 中定义 MambaSlotAllocator，管理空闲插槽列表，提供 alloc、free、clear 及批处理接口 alloc_group_begin/end。
2. 剥离 MambaPool 的分配职责：从 memory_pool.py 中移除 free_slots 初始化及所有分配/释放方法，改为持有 MambaSlotAllocator 实例并委托。
3. 更新调用方：在所有引用 req_to_token_pool.mamba_pool 的地方（调度器 scheduler.py、Mamba 缓存组件 mamba_component.py、mamba_radix_cache.py、hi_mamba_radix_cache.py、invariant_checker.py、pool_stats_observer.py、schedule_batch.py、streaming_session.py 等）替换为 mamba_allocator。
4. 调整测试：更新 test_mamba_unittest.py 以适应新接口。

关键文件：

- python/sglang/srt/mem_cache/allocator/mamba.py（模块 分配器；类别 source；类型 core-logic；符号 MambaSlotAllocator, init, available_size, alloc_group_begin）：新增核心分配器类，所有 Mamba 插槽分配最终委托至此。
- python/sglang/srt/mem_cache/memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 available_size, alloc_group_begin, alloc_group_end, alloc）：主内存池文件，从中剥离了 Mamba 分配相关代码，是重构的关键目标。
- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic）：调度器中调度循环直接调用分配器，是主要调用方之一。
- test/registered/unit/mem_cache/test_mamba_unittest.py（模块 测试；类别 test；类型 test-coverage）：测试用例适配新接口，确保重构正确性。

- python/sglang/srt/mem_cache/unified_cache_components/mamba_component.py（模块 统一缓存；类别 source；类型 core-logic）：统一缓存组件中 Mamba 状态管理使用分配器，修改了分配 / 释放路径。

关键符号：MambaSlotAllocator.init, MambaSlotAllocator.available_size, MambaSlotAllocator.alloc_group_begin, MambaSlotAllocator.alloc_group_end, MambaSlotAllocator.alloc, MambaSlotAllocator._do_alloc, MambaSlotAllocator.free, MambaSlotAllocator.clear

关键源码片段

python/sglang/srt/mem_cache/allocator/mamba.py

新增核心分配器类，所有 Mamba 插槽分配最终委托至此。

```
# 管理 Mamba 池空闲插槽列表的分配器类
class MambaSlotAllocator:
    def __init__(self, size: int, device: str):
        self.size = size
        self.device = device
        self._alloc_iter = None # 批分配游标，初始为 None
        self.clear()

    def available_size(self) -> int:
        return len(self.free_slots)

    def alloc_group_begin(self, num_reqs: int):
        # 预分配一批插槽以减少 match_prefix 过程中的分配开销
        self._alloc_iter = None
        if num_reqs > 0:
            result = self._do_alloc(num_reqs)
            if result is not None:
                self._alloc_iter = iter(result.split(1))

    def alloc_group_end(self):
        # 返还组内未使用的预分配插槽到空闲列表
        if self._alloc_iter is not None:
            remaining = list(self._alloc_iter)
            if remaining:
                self.free(torch.cat(remaining))
            self._alloc_iter = None

    def alloc(self, need_size: int) -> Optional[torch.Tensor]:
        # 如果处于批分配窗口且请求 size 为 1，优先使用预分配游标
        if self._alloc_iter is not None and need_size == 1:
            slot = next(self._alloc_iter, None)
            if slot is not None:
                return slot
        return self._do_alloc(need_size)
```

```
    def _do_alloc(self, need_size: int) -> Optional[torch.Tensor]:
        if need_size > len(self.free_slots):
            return None
        select_index = self.free_slots[:need_size]
        self.free_slots = self.free_slots[need_size:]
        return select_index

    def free(self, free_index: torch.Tensor):
        if free_index.numel() == 0:
            return
        self.free_slots = torch.cat((self.free_slots, free_index))

    def clear(self):
        # Slot 0 保留给填充 token 的 dummy 写入目标
        self.free_slots = torch.arange(
            1, self.size + 1, dtype=torch.int64, device=self.device
        )
```

评论区精华

- Gemini Code Assist 建议：free_group_end 未清空 _free_group 可能导致 GPU 内存泄漏；free 方法中应显式将 free_index 移至 self.device 以避免设备不匹配。
- Codex 提示：HiMambaRadixCache 中的 _alloc_with_evict 仍引用 mamba_pool.alloc，若 HiCache 路径启用将会 Crash。
- ispobock：建议保留部分原有注释。作者已采纳。
  - free_group_end 内存泄漏 (correctness): 最终代码中 _free_group 已被移除，该问题已规避。
  - free 方法设备不匹配 (correctness): 未采纳，但当前调用方保证张量在同一设备，风险较低。
  - HiMambaRadixCache 分配路径未迁移 (correctness): 该 PR 未完全迁移 HiCache 路径，后续 PR 可能解决。

风险与影响

- 风险：
  1. HiMamba 路径不完整：hi_mamba_radix_cache.py 中仅修改了 _free_device_mamba 一处，其他分配路径（如 COW、load-back）仍使用 mamba_pool，若启用 HiCache 将引发 AttributeError。
  2. 设备不匹配：MambaSlotAllocator.free 未对输入张量做设备转换，若调用方传入 CPU 张量可能导致 torch.cat 异常。
  3. 回归风险：重构涉及 18 个文件，虽然测试通过，但 Mamba 功能路径未在 CI 中充分覆盖（Mamba 模型较少）。 - 影响：影响范围：使用 Mamba 状态空间模型（如 Mamba2）的场景。重构后接口从 mamba_pool 变为 mamba_allocator，但所有调用方已在本 PR 中更新，外部无感知。代码组织更清晰，分配器与存储解耦，便于后续扩展。 - 风险标记：HiCache 分配路径未完全迁移，设备不匹配潜在风险，Mamba 功能测试覆盖不足

关联脉络

- PR #25371 [RFC][Refactor] mem_cache pool / allocator restructure: 本 PR 为此 RFC 的子任务 (4/N)。
- PR #26675 split allocator.py into allocator/ subpackage: 同一重构系列的第一步，提取基础分配器架构。
- PR #26676 SWATokenToKVPoolAllocator: 第二步，类似地提取 SWA 分配器。