PR #25000 完整报告

sgl-project/sglang

Reduce mamba prefill allocation overhead

合并时间：2026-06-04 15:10

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/25000

执行摘要

- 一句话：降低 Mamba 预填充内存分配开销
- 推荐动作：该 PR 值得精读，尤其是 MambaPool 分组分配的设计模式，可以推广到其他类似的热点分配路径。关注 alloc_group_end 的提前释放逻辑和与现有 free 调用的交互。建议添加单元测试覆盖分组分配的正确性（例如分配后释放、迭代器耗尽回退等场景）。

功能与动机

Reduce CPU overhead in prefill scheduling by batching mamba state slot allocation. The scheduler opens one grouped allocation window for the waiting queue, and normal mamba_pool.alloc(1) calls draw from that preallocated batch instead of repeatedly doing per-request allocation bookkeeping.

实现拆解

1. MambaPool 新增分组分配接口：在 python/sglang/srt/mem_cache/memory_pool.py 的 MambaPool 类中添加 alloc_group_begin(num_reqs) 和 alloc_group_end() 方法。alloc_group_begin 一次性分配 num_reqs 个 slot，并将结果拆分成单 slot 迭代器保存在 _alloc_iter 实例变量中。alloc_group_end 释放未使用的预分配 slot。
2. 修改 alloc 方法：当 _alloc_iter 非空且请求大小为 1 时，直接从迭代器中获取下一个 slot；否则回退到原有 _do_alloc 路径（抽取为单独方法）。这样对现有单次分配调用透明，仅在分组窗口内生效。
3. 调度器集成：在 python/sglang/srt/managers/scheduler.py 的 _get_new_batch_prefill_raw 中，遍历等待队列前获取 mamba_pool 并调用 alloc_group_begin(len(self.waiting_queue))，遍历结束后调用 alloc_group_end()，确保未使用的预分配 slot 被释放。
4. 类型导入调整：在 memory_pool.py 的 typing 导入中添加 Iterator，以支持 Optional[Iterator] 类型标注。

关键文件：

- python/sglang/srt/mem_cache/memory_pool.py（模块 缓存层；类别 source；类型 core-logic；符号 alloc_group_begin, alloc_group_end, _do_alloc）：核心变更文件。实现 MambaPool 分组分配接口 alloc_group_begin/alloc_group_end，修改 alloc 方法以优先从预分配迭代器中获取 slot，抽取 _do_alloc 作为底层分配逻辑。

- python/sglang/srt/managers/scheduler.py（模块 调度器；类别 source；类型 core-logic）：调度器集成。在 _get_new_batch_prefill_raw 中，遍历等待队列前调用 alloc_group_begin(len(self.waiting_queue))，遍历后调用 alloc_group_end()，实现分组分配在调度流程中的包裹。

关键符号：alloc_group_begin, alloc_group_end, alloc, _do_alloc

关键源码片段

python/sglang/srt/mem_cache/memory_pool.py

核心变更文件。实现 MambaPool 分组分配接口 alloc_group_begin/alloc_group_end，修改 alloc 方法以优先从预分配迭代器中获取 slot，抽取 _do_alloc 作为底层分配逻辑。

```
# 从 typevars 中新增 Iterator 导入用于类型标注
from typing import TYPE_CHECKING, Any, Iterator, List, Optional, Tuple, Union

class MambaPool:
    def __init__(self, ...):
        # ... 原有初始化逻辑 ...
        # Active preallocated batch for `alloc_group_begin` / `alloc_group_end`.
        # When non-None, `alloc(1)` consumes the next slot from this iterator
        # instead of calling `_do_alloc(1)` per request. Reset to None outside
        # a group window so `alloc` falls through to the per-call path.
        self._alloc_iter: Optional[Iterator] = None

    def alloc_group_begin(self, num_reqs: int):
        """开始分组分配。预先分配 num_reqs 个 slot，并将其拆分为单 slot 迭代器。"""
        self._alloc_iter = None # 重置，确保不会残留旧迭代器
        if num_reqs > 0:
            result = self._do_alloc(num_reqs)
            if result is not None:
                # 将结果张量沿第一维拆分成单个 slot，创建迭代器
                self._alloc_iter = iter(result.split(1))

    def alloc_group_end(self):
        """结束分组分配，释放未使用的预分配 slot。"""
        if self._alloc_iter is not None:
            # 收集剩余未使用的 slot
            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]:
        """分配 slot。若在分组窗口内且请求大小为 1，则从预分配迭代器中获取。"""
        if self._alloc_iter is not None and need_size == 1:
            # 优先使用预分配 slot
            slot = next(self._alloc_iter, None)
            if slot is not None:
```

```
                return slot
            # 迭代器可能已用尽，回退到底层分配
        return self._do_alloc(need_size)

    def _do_alloc(self, need_size: int) -> Optional[torch.Tensor]:
        """底层分配逻辑，从 free_slots 中切片。"""
        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
```

评论区精华

核心讨论是 _alloc_iter 的初始化位置。ispobock 在 review 中建议将 _alloc_iter 的声明移至 __init__ 并添加注释，以便清晰表达其生命周期。开发者根据建议进行了修改（commit 1511004），移除了 alloc_group_begin 内的局部类型标注，改为在 __init__ 中初始化为 None 并添加文档注释。

- _alloc_iter 初始化位置 (design): 开发者接受建议，在第二版提交中将 _alloc_iter 初始化为 None 并添加文档注释，同时移除了 alloc_group_begin 中的局部类型标注。

风险与影响

- 风险：
  1. 内存泄漏: 如果 alloc_group_end 未正确调用（例如异常路径），预分配的 slot 不会被释放，导致可用 Mamba 缓存减少。当前代码仅在调度器遍历循环的 finally 等效位置（break/ 正常结束）调用 alloc_group_end，覆盖了正常和中断路径，风险较低。
  2. 并发安全: MambaPool 通常运行在单线程调度器中，没有锁；但若未来引入多线程调度，_alloc_iter 的读写可能产生竞态。目前无此场景。
  3. 与 COW 策略的交互: PR body 明确指出不推迟 COW 复制（copy_from 立即执行），避免了因延迟复制导致源 slot 被回收的风险。 - 影响：该 PR 直接影响调度器的预填充阶段，对使用 Mamba 模型（如 Mamba2、Mamba3）的场景降低 CPU 分配开销。影响范围仅限于启用了 Mamba 缓存的模型，对其他模型（纯 Transformer、混合 MoE 等）无影响。改动量小（+35/-1），代码结构清晰，回退容易。 - 风险标记：缺少测试覆盖

关联脉络

- 暂无明显关联 PR