PR #28612 完整报告

sgl-project/sglang

Optimize C128 state pool allocation using request state pool

合并时间：2026-07-01 10:11

原文链接：http://prhub.com.cn/sgl-project/sglang/pull/28612

执行摘要

- 一句话：优化 C128 状态池分配，解耦 SWA 映射修复精度
- 推荐动作：该 PR 值得精读，因为它展示了如何解决 KV 缓存管理中的生命周期不匹配问题，以及如何从显式 SWA 映射过渡到请求级状态分配。对于理解 DeepSeek-V4 的压缩状态管理和 PD 分离架构很有价值。重点关注 pool_configurator.py 中内存计算调整、deepseek_v4_memory_pool.py 中状态布局重构以及新的 JIT kernel 清理机制。

功能与动机

当多轮请求命中 radix-cached 前缀时，全 KV 前缀仍在 radix 树中存活，但对应的 SWA 映射可能已被清除或重用。在线 C128/MTP 路径通过 full_to_swa_index_mapping 和 swa_page_size 定位状态槽，若 SWA 映射被释放，则可能读取槽 0、旧槽或重用槽，导致精度下降。此 PR 通过将 C128 状态索引与 SWA 映射解耦来解决此问题。

实现拆解

1. 解耦 C128 状态索引：在 deepseek_v4_memory_pool.py 中，C128 状态池的大小计算不再依赖 SWA 映射，改为请求级固定分配；online 路径直接使用 req_pool_idx 索引，offline 路径使用 req_pool_idx * ring_size + position % ring_size 环形缓冲。
2. 修改池配置模型：在 pool_configurator.py 中，_get_bytes_per_full_token 移除 C128 状态按 token 比例缩放，新增 _get_c128_state_fixed_bytes 和 _get_num_req_slots 方法，并在 _compute_dsv4_sizes 中引入 finalize_with_max_running_requests 以在约束路径下重新计算固定内存。
3. 新增 JIT Kernel 清理：新增 c128_cleanup.py，实现 Triton kernel _clear_unaccepted_c128_draft_states_kernel，用于高效清零未接受草稿状态。
4. 适配 PD 分离传输：在 disaggregation/utils.py 中新增 get_dsv4_c128_state_indices 函数，在 disaggregation/prefill.py 和 disaggregation/decode.py 中新增 _c128_state_payload 以传递请求级状态索引。
5. 适配压缩路径：在 compress_hip.py 和 compressor.py 中，C128 状态定位从 SWA 映射改为直接通过 translate_from_req_position_to_state_loc 计算。
6. 测试配套：新增 TestDSV4C128StateIndices 单元测试，覆盖 online/offline 边界情况。

关键文件：

- python/sglang/srt/model_executor/pool_configurator.py（模块 内存配置；类别 source；类型 data-contract；符号 finalize_with_max_running_requests, _get_num_req_slots, _get_c128_state_fixed_bytes, _get_c128_state_fixed_bytes_for_token_capacity）：核心配置变更，调整 C128 状态池计算模型，引入 finalize_with_max_running_requests 和请求级固定内存估算
- python/sglang/srt/mem_cache/deepseek_v4_memory_pool.py（模块 内存池；类别 source；类型 core-logic；符号 get_c128_state_buf_infos, get_online_c128_state_num_req_slots, clear_c128_req_state, clear_unaccepted_c128_draft_states）：核心运行时变更，实现 C128 状态索引解耦，新增 get_c128_state_buf_infos 和清理函数
- python/sglang/jit_kernel/dsv4/c128_cleanup.py（模块 JIT 内核；类别 source；类型 core-logic；符号 _clear_unaccepted_c128_draft_states_kernel, clear_unaccepted_c128_draft_states）：新增 JIT kernel，实现未接受草稿状态的快速清理，提升 MTP 效率
- python/sglang/srt/disaggregation/utils.py（模块 分离传输；类别 source；类型 core-logic；符号 is_dsv4_c128_online_enabled, get_dsv4_c128_state_indices）：新增 C128 状态索引辅助函数，用于 PD 分离传输
- python/sglang/srt/disaggregation/prefill.py（模块 分离传输；类别 source；类型 core-logic；符号 _c128_state_payload）：适配 PD 预填充端发送 C128 状态 payload

关键符号：finalize_with_max_running_requests, _get_num_req_slots, _get_c128_state_fixed_bytes, _get_c128_state_fixed_bytes_for_token_capacity, get_c128_state_buf_infos, get_online_c128_state_num_req_slots, clear_c128_req_state, clear_unaccepted_c128_draft_states, is_dsv4_c128_online_enabled, get_dsv4_c128_state_indices, _c128_state_payload

关键源码片段

python/sglang/srt/model_executor/pool_configurator.py

核心配置变更，调整 C128 状态池计算模型，引入 finalize_with_max_running_requests 和请求级固定内存估算

```
class DSV4PoolConfigurator(MemoryPoolConfigurator):
    # ... (init collecting fields) ...

    def finalize_with_max_running_requests(
        self, config: MemoryPoolConfig
    ) -> MemoryPoolConfig:
        """在约束路径下，根据 max_running_requests 重新计算 C128 状态池大小。"""
        if config.max_running_requests is not None:
            num_req_slots = self._get_num_req_slots(
                config.max_running_requests)
            fixed_bytes = self._get_c128_state_fixed_bytes()
            # 计算 C128 状态占用的 token 等价空间
            c128_state_tokens = ceil_div(
```

```
                fixed_bytes * num_req_slots,
                self._get_bytes_per_full_token(),
            )
            # 从总 token 中减去 C128 固定开销
            adjusted_max_total = config.max_total_num_tokens - c128_state_tokens
            # 重新计算池大小（调用 _compute_dsv4_sizes 重新分配）
            return self._compute_dsv4_sizes(adjusted_max_total, self.page_size)
        return config

    def _get_num_req_slots(self, max_running_requests: int) -> int:
        """根据 max_running_requests 和 DP 并行度计算请求槽位数。"""
        per_worker = max_running_requests // self.dp_size
        # PD decode 需要额外槽位用于预传输
        if self.disaggregation_mode == "decode":
            per_worker += self.disaggregation_decode_extra_slots
        return per_worker

    def _get_c128_state_fixed_bytes(self) -> int:
        """返回每个请求槽位固定的 C128 状态字节数。
        Online: 仅存储 1 组 (max, sum, kv) 状态，即 C128 环大小=1。
        Offline: 存储完整 128 槽原始状态。
        """
        state_dtype_size = torch.tensor([], dtype=self.c128_state_dtype).element_size()
        if self.c128_online:
            # online: 每请求 1 组状态，包含 2*head_dim 的 max/sum 和 head_dim*kv 的 kv
            # 具体数值由 C128 层数和头维决定
            bytes_per_req = self.num_layers_ca128 * (
                2 * self.indexer_head_dim * state_dtype_size # max+sum
                + (self.qk_nope_head_dim + self.qk_rope_head_dim * 2) * state_dtype_size # kv
            )
        else:
            # offline: 每请求 128 组原始 token 状态
            bytes_per_req = (
                128 * self.num_layers_ca128
                * (self.qk_nope_head_dim + self.qk_rope_head_dim * 2)
                * state_dtype_size
            )
        return bytes_per_req

    def _get_c128_state_fixed_bytes_for_token_capacity(self) -> int:
        """用于 token 容量估算时的固定开销（与请求数无关的常量部分）。"""
        # C128 状态现在是请求级，不在 token 容量估算中，返回 0
        return 0
```

python/sglang/srt/mem_cache/deepseek_v4_memory_pool.py

核心运行时变更，实现 C128 状态索引解耦，新增 get_c128_state_buf_infos 和清理函数

```
class DeepSeekV4TokenToKVPool(BaseSWAKVPool):
    def __init__(self, ..., c128_state_pool_size, ...):
```

```
        # ...
        c128_ring_size = self.get_ring_size(128)
        if ONLINE_C128:
            # Online: 状态池大小按请求槽位数（num_req_slots）分配
            # 每个请求固定 1 组状态，索引直接使用 req_pool_idx
            c128_state_pool_size = max(c128_state_pool_size, self.num_req_slots)
        else:
            # Offline: 每个请求保持 raw state ring，大小为 num_req_slots * ring_size
            c128_state_pool_size = max(
                c128_state_pool_size, self.num_req_slots * c128_ring_size
            )
        self.c128_state_pool_size = c128_state_pool_size
        # 记录实际用于 online C128 的请求槽数（供 MTP 序列长度数组用）
        self.online_c128_state_num_req_slots = c128_state_pool_size
        # ...

    def get_c128_state_buf_infos(self):
        """返回所有 C128 状态缓冲区的指针、字节大小和条目大小列表，用于 PD 传输。"""
        data_ptrs, data_lens, item_lens = [], [], []
        for pool in self.compress_state_pools:
            if pool is None or pool.ratio != 128:
                continue
            t = pool.kv_score_buffer.kv_score
            assert t.ndim == 2
            data_ptrs.append(t.data_ptr())
            data_lens.append(t.nbytes)
            # online 模式下，每个条目是单组状态（1 行）；offline 是 128 行
            item_lens.append(t[0].nbytes if ONLINE_C128 else t[0].nbytes * 128)
        return data_ptrs, data_lens, item_lens

    def clear_c128_req_state(self, req_pool_idx: int):
        """清零指定请求槽位的 C128 状态（在请求分配时调用）。"""
        for pool in self.compress_state_pools:
            if pool is None or pool.ratio != 128:
                continue
            if ONLINE_C128:
                # online: 只清零一行
                pool.kv_score_buffer.kv_score[req_pool_idx].zero_()
            else:
                # offline: 清零整个 ring
                ring_start = req_pool_idx * self.get_ring_size(128)
                ring_end = ring_start + self.get_ring_size(128)
                pool.kv_score_buffer.kv_score[ring_start:ring_end].zero_()
```

python/sglang/jit_kernel/dsv4/c128_cleanup.py

新增 JIT kernel，实现未接受草稿状态的快速清理，提升 MTP 效率

```
import torch
import triton
```

```
import triton.language as tl


@triton.jit
def _clear_unaccepted_c128_draft_states_kernel(
    state, # [num_req_slots * ring_size, 2 * half] 状态张量
    req_pool_indices, # [batch_size] 每个请求的池索引
    seq_lens, # [batch_size] 当前序列长度
    accept_lens, # [batch_size] 各请求已接受的草稿长度
    ring_size: tl.constexpr, # per-request ring 大小（online=1，offline=128）
    half: tl.constexpr, # 状态后半部分的起始索引（用于填充 -inf）
    num_draft_tokens: tl.constexpr, # 最大草稿数
    BLOCK_D: tl.constexpr, # 每块处理的维度
):
    bid = tl.program_id(0) # 哪个请求
    draft_offset = tl.program_id(1) # 哪个草稿位置
    block_id = tl.program_id(2) # 维度块

    accept_len = tl.load(accept_lens + bid)
    # 如果该草稿位置已被接受，跳过清理
    if draft_offset < accept_len:
        return

    req_pool_idx = tl.load(req_pool_indices + bid).to(tl.int64)
    seq_len = tl.load(seq_lens + bid).to(tl.int64)
    # 计算在 ring 中的槽位
    slot = (seq_len + draft_offset) % ring_size
    row = req_pool_idx * ring_size + slot

    offsets = block_id * BLOCK_D + tl.arange(0, BLOCK_D)
    mask = offsets < half
    row_base = row * (half * 2)
    # 前半部分置 0（max/sum），后半部分置 -inf（表示无效）
    tl.store(state + row_base + offsets, 0.0, mask=mask)
    tl.store(state + row_base + half + offsets, float("-inf"), mask=mask)


def clear_unaccepted_c128_draft_states(
    state: torch.Tensor,
    req_pool_indices: torch.Tensor,
    seq_lens: torch.Tensor,
    accept_lens: torch.Tensor,
    *,
    ring_size: int,
    num_draft_tokens: int,
) -> None:
    half = state.shape[-1] // 2
    # 启动三维网格：请求 x 草稿位置 x 维度块
    _clear_unaccepted_c128_draft_states_kernel[
```

```
        (req_pool_indices.numel(), num_draft_tokens, triton.cdiv(half, 256))
    ](
        state,
        req_pool_indices,
        seq_lens,
        accept_lens,
        ring_size,
        half,
        num_draft_tokens,
        BLOCK_D=256,
    )
```

评论区精华

1. C128 状态定位常量的含义：DarkSharpness 质疑代码中固定 128 的含义，作者解释这是 full_loc / 128 的 C128 槽位映射，而非 SWA 页大小。
2. Radix 树不需要缓存 C128 状态：ispobock 指出 DSV4 页大小是 256，C128 状态在 radix 中不适用，作者随后移除了 radix 节点中的 C128 快照 / 恢复逻辑，仅保留复位。
3. transfer_input_len 传递争议：ShangmingCai 提出简化传输长度传递，作者解释在 create_sender 时 fill_len 可能还未生效，最终采用在 finalize_bootstrap 中设置 transfer_input_len。
4. 精度测试要求：ispobock 要求更多 AIME25 重复测试（>16 次），作者补充了精度数据。
5. 环境变量回滚保护：ispobock 询问是否有环境变量用于回滚，作者表示该 PR 是向前兼容的，但无专门回滚开关。
6. 代码风格：merrymercy 指出不应使用 getattr，作者确认修复。

- C128 状态定位中的常量 128 含义 (question): 作者确认是每 128 个全 KV token 对应一个 C128 状态槽，与 SWA 页大小无关。
- Radix 树不需要缓存 C128 状态 (design): 作者移除了 radix 节点中的 C128 快照 / 恢复逻辑，仅保留在 radix 缓存命中时的复位操作。
- transfer_input_len 传递方式简化 (design): 采用在 finalize_bootstrap 中设置 transfer_input_len = len(req.origin_input_ids)，并移除 decode 端的额外传递。
- 精度测试要求 (testing): 作者补充了更详细的精度测试数据，包括多次重复的 pass@1 均值以及 SEM。

风险与影响

- 风险：
  1. 回归风险：32 个文件、33 次提交，大量代码变动可能引入新 bug，尤其是 PD 传输路径改动和配置文件计算模型调整。
  2. 性能风险：C128 状态池从 token 比例改为请求级分配，在极高并发（请求数接近 max_running_requests）时内存利用率可能下降，但测试显示无明显退化。
  3. 兼容性风险：新 JIT kernel c128_cleanup.py 使用 Triton，在 AMD/Intel 架构上可能需额外适配（当前已有 _IS_HIP 检查，但仅在非 AMD 上启用）。

4. PD 分离风险：新增 C128_STATE StateType，若 decode 端未正确更新，可能导致状态传输不完整。
5. Radix 缓存复位时机：在 radix 缓存命中时复位 C128 状态，如果复位逻辑不精确，可能丢失必要状态。 - 影响：影响范围：仅影响 DeepSeek-V4 模型中使用在线 C128 压缩（SGLANG_OPT_USE_ONLINE_COMPRESS=1）的场景。修复了多轮请求中精度异常的问题，同时降低了 C128 状态内存占用（从 token 比例变为请求数比例）。影响程度：核心路径变更，但精度测试证明修复有效；PD 分离用户需同步更新 decode 端。 - 风险标记：核心路径变更，新 JIT Kernel 跨平台兼容，PD 传输改动影响分离部署，内存分配模型改变，大量代码变动增加回归风险

关联脉络

- PR #31163 Extract per-architecture KV-cache pool builders into KVCacheConfigurator: 涉及相同模块 pool_configurator.py 的内存池配置重构，本 PR 的 C128 状态池配置调整与其相关。
- PR #30937 fix: avoid double KV release on disaggregated prefill grammar errors: 同为解耦预填充修复，涉及 KV 生命周期管理，与本 PR 的 C128 状态生命周期修复有相似背景。